```perl
#!/usr/local/bin/perl
#
# given a date, this will count how many messages are in the spam database
# that were inserted on that date
use strict;
use MiaVia::PipelineSettings;
use MiaVia::PipelineUtils;
use MiaVia::Utils;
use MiaVia::DbUtil;
my ($first_date) = @ARGV;
my $count = 0;
MiaVia::PipelineSettings->initialize(); # initialize configurations
my ($fyear, $fmon, $fday) = split "-", $first_date;
my $today = "$first_date 00:00:00";
my $next_day = $fday + 1;
my $tomorrow = "$fyear-$fmon-$next_day 00:00:00";
#print "today: $today tomorrow $tomorrow";
my $dbh = MiaVia::DbUtil::openDb('miavia_rdr');
my $queryStr = qq{ select count(*) FROM Message
 WHERE Message_inserted_on > '$today' and Message_inserted_on < '$tomorrow'};
my $sth = $dbh->prepare($queryStr);
my $ret = $sth->execute();
($count) = $sth->fetchrow_array();
$sth->finish();


print "$count messages are in the database that were inserted on $first_date\n";
```

```
#!/usr/local/bin/perl
use strict;
my $count = 0;
my ($indir, $outdir) = @ARGV;
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
if (!($outdir =~/\/$/)) {
 $outdir = $outdir.'/';
}
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    while (my $line = <IN>) {
        if (

($line =~ /\@yahoogroups.com/) ||

#         add more conditional lines below this line
####################################################
($line =~ /woodyswatch.com/) ||
($line =~ /\.ebay.com/) ||
($line =~ /\.paypal.com/) ||
($line =~ /Freaky Freddies/) ||
($line =~ /PSYPHY/) ||
($line =~ /sdforum.org/) ||
($line =~ /topica.com/) ||
($line =~ /Original message follows/) ||
($line =~ /ProMED Digest/) ||
($line =~ /Newsletters.Microsoft.com/) ||
($line =~ /eBay Item/) ||
($line =~ /order is on its way/) ||
($line =~ /Microsoft .NET Passport/) ||
($line =~ /MedPulse is a weekly/) ||
($line =~ /\.vanguard.com/) ||
($line =~ /ticketmaster.com/) ||
($line =~ /MOMYS Digest/) ||
($line =~ /Wired News Daily/) ||
($line =~ /e-ProTalk/) ||
($line =~ /www.frontlinethoughts.com/) ||
($line =~ /BusinessWire.com/) ||
($line =~ /\.webmd.com/) ||
($line =~ /The Daily Reckoning/) ||
($line =~ /McAfee Dispatch/) ||
($line =~ /www.m0.net/) ||
($line =~ /Message From Fastclick Contact Form/) ||
($line =~ /Stockhouse Newsblast/) ||
($line =~ /www.kintera.org/) ||
($line =~ /addresses had permanent fatal/) ||
($line =~ /\.responsys.com/) ||
($line =~ /\.rsc03.net/) ||
($line =~ /info.drugstore.com/) ||
($line =~ /\.namesdatabase.com/) ||
#($line =~ //) ||
# To add new strings to look for,
# copy this line below, and place the new line somewhere between the first and last
# conditional lines (they are special), then, add your string, without any
# punctuation characters, between the two /s, and remove the leading "#" to
# uncomment the line, and you are good to go
#
# if you need to use the following chars in your string, preceed them with
# a backslash: # / @ ? & ! = .   So, a string like "gal@yahoo.com" would need to become
#  "gal\@yahoo\.com"
# note, in a comparison "." stands for any single character, so when it is
# important that it actually *is* a ".", you should back slash it.
#
# below are some blank conditional lines, commented out.
# ($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#
##################################################
# add more conditional lines above this line
($line =~ /\.linksynergy.com/)
# actual string is ""
) { # we found a probable non-spam
        $count++;
        rename  "$indir$fname","$outdir$fname"
         or die ("can't rename file $indir$fname to $outdir$fname: $!");
```

```
  goto DONE;
      }  # end if
    } # end while still lines in file
    DONE:
    close IN;
} # end foreach
# print "$count files moved\n";
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2994
#                       All Rights Reserved
#  $Id: Message.pm,v 1.31 2004/06/15 02:47:18 lizd Exp $
#
package MiaVia::Message;
## A Message object represents a "processed" email message,
## which can be inserted into the db (if known to be spam)
## or tested against the db (to determine its spamicity), etc.
##
## modified 07-25-02 by Liz Derr - added getHeaderTag method
## modified 9-27-03 by Liz Derr - added path to message
## modified 11-23-03 by LD - added getMD5 from code contributed by Joe Harris
use Digest::MD5 qw(md5_hex);
use strict;
use MiaVia::SysLogEvent qw(debug_msg);
## CLASS METHODS
##
# Create a Message object, given
#
sub new {
  my ($class, $version, $fingerSet, $optArgs) = @_;
  # required and defaults
  my $self = {  version => $version
             ,  fingerSet => $fingerSet
             ,  fileName => ""
             ,  path => ""    # added 9-27-03
             ,  category => 1 # category of 1 = unreviewed
             };
  # optional and overriding defaults
  for my $arg ( 'category'
              , 'fileName'
              , 'mimeProcessor'
              , 'smtpEnvelope'
              , 'recipient'
              , 'path') # added 9-27-03
  {
    if (exists $optArgs->{$arg}) {
      $self->{$arg} = $optArgs->{$arg};
    }
  }
  bless $self, $class;
}
## INSTANCE METHODS
##
# Getters
# get the Message object's category
sub category {
  my $self = shift;
  return $self->{category};
```

```
}
# get the Message object's fileName
#
sub fileName {
  my $self = shift;
  return $self->{fileName};
}
# get the Message object's recipient (the username in the To:)
#
sub recipient {
  my $self = shift;
  return $self->{recipient};
}
# get the file's path name
#
sub path {
  my $self = shift;
  return $self->{path};
}
# get the Message object's FingerSet
#
sub fingerSet {
  my $self = shift;
  return $self->{fingerSet};
}
# get the Message object's SMTP envelope
#
sub smtpEnvelope {
  my $self = shift;
  return $self->{smtpEnvelope};
}
# get the Message object's version
#
sub version {
  my $self = shift;
  return $self->{version};
}
# get the Message object's fingers' digests
#
sub allDigests {
  my $self = shift;
  return $self->{fingerSet}->allDigests();
}
# MIME Processor Methods
# add message header
#
sub addHeader {
  my ($self, $tag, $text) = @_;
  my $mp = $self->{mimeProcessor};
  if (defined $mp) {
   $mp->addHeader($tag, $text);
  } else {
   debug_msg("mimeProcessor not defined, can't add header tag");
  }
}
# add unique message header
#
sub addUniqueHeader {
  my ($self, $tag, $text) = @_;
  my $mp = $self->{mimeProcessor};
  if (defined $mp) {
   $mp->addUniqueHeader($tag, $text);
  } else {
   debug_msg("mimeProcessor not defined, can't add header tag");
  }
}
# add unique message
#
sub prependToHeader {
  my ($self, $tag, $text) = @_;
  my $mp = $self->{mimeProcessor};
  if (defined $mp) {
   $mp->prependToHeader($tag, $text);
  } else {
   debug_msg("mimeProcessor not defined, can't prepend header tag");
  }
}
# get header tag
#
sub getHeaderTag {
  my ($self, $tag) = @_;
  my $text;
  my $mp = $self->{mimeProcessor};
  if (defined $mp) {
   $text = $mp->getHeaderTag($tag);
  } else {
   debug_msg("mimeProcessor not defined, can't get header tag");
  }
  return $text;
}
# get length
# calculates the content-length in bytes of the entire message and retruns it.
#
sub getLength {
  my ($self) = @_;
  my $mp = $self->{mimeProcessor};
  my $text;
  if (defined($mp)) {
   $text = $mp->{entity}->as_string;
  } else {
   debug_msg("mime processor not defined");
   return(0);
  }
  return (length($text));
}
# get length of all fingers in message
#
sub getFingLengthTotal {
  my ($self) = @_;
  return ($self->{fingerSet}->allFingerLengths());
}
# clean up the MIME Processor
#
sub cleanup {
  my $self = shift;

  my $mp = $self->{mimeProcessor};
  $mp->cleanup();
}
# print MIME/SMTP messages to output stream
#
sub printMimeTo {
  my ($self, $outStream) = @_;
  my $mp = $self->{mimeProcessor};
  $mp->printTo($outStream);
}
sub printSmtpTo {
  my ($self, $outStream) = @_;
  if (exists $self->{smtpEnvelope}) {
    print $outStream $self->{smtpEnvelope};
    print $outStream "DATA\n";
    my $mp = $self->{mimeProcessor};
    $self->printMimeTo($outStream);
    print $outStream ".\n"
  }
}
# re-digest the fingerset
#
sub redigest {
  my $self = shift;
  my $fingerSet = $self->{fingerSet};
  $fingerSet->redigest();
}
# render the Message object as an XML element
#
#sub xmlElem {
#  my $self = shift;
 # my $messageElem = XML::LibXML::Element->new('Message');
  #my $fingerSetElem = $self->{fingerSet}->xmlElem();
  #$messageElem->appendChild($fingerSetElem);
 # return $messageElem;
#}
#sub xmlString {
  #my $self = shift;
  #return $self->xmlElem->toString();
#}
# added 11-23-03 from Joe Harris' code
sub getMD5 {
 my ($self) = @_;
 my $mp = $self->{mimeProcessor};
 my $text = $mp->{entity}->as_string;
 my ($headers,@body) = split(/\n\n/,$text);
 my $bodytext = join("\n\n",@body);
 my $md5hash = md5_hex($bodytext);
 return $md5hash;
}
1; # end of Message.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: MessageSource.pm,v 1.14 2004/06/04 02:28:23 lizd Exp $
#
# modified 09-03-02 by Liz Derr added call to call_for_help in place of die
# 2-15-04 clear of Settings references
# 2-16-04 removed support for symbolic dir
# 6-3-04 removed LibXML
package MiaVia::MessageSource;
## MessageSource specifies where the data to create a Message
## comes from.
use strict;
#use XML::LibXML;
use MiaVia::Utils;
## CLASS METHODS
##
# Construct a new MessageSource object, given
# a hash containing a value for one of the listed
# message source types
#
sub new {
  my ($class, $type, $optArgs) = @_;
  do {
    warn ("Type is $type optArgs is $optArgs - Cannot create MessageSource!\n");
  } unless ($type && $optArgs);
  my $self = { type => $type };
  if ($type eq 'mimeFile') {
    do {
        warn ("Cannot create MessageSource of type 'mimeFile'\n");
    } unless ((exists $optArgs->{fileName})
            && (exists $optArgs->{fileExtension})
            && (exists $optArgs->{actualDir}));
    $self->{fileName} = $optArgs->{fileName};
    $self->{fileExtension} = $optArgs->{fileExtension};
    $self->{dir} = $optArgs->{actualDir};
  } elsif ($type eq 'mimeString') {
    do {
         warn ( "Cannot create MessageSource of type 'mimeString'");
    } unless (exists $optArgs->{mimeString});
    $self->{mimeString} = $optArgs->{mimeString};
  } elsif ($type eq 'smtp') {
    do {
         warn ( "Cannot create MessageSource of type 'smtp'"
                . " without SMTP envelope, directory, MIME file"
                . " name and extension\n");
    } unless ((exists $optArgs->{smtpEnvelope})
            && (exists $optArgs->{fileName})
            && (exists $optArgs->{fileExtension})
            && (exists $optArgs->{actualDir}));
    $self->{smtpEnvelope} = $optArgs->{smtpEnvelope};
    $self->{fileName} = $optArgs->{fileName};
    $self->{fileExtension} = $optArgs->{fileExtension};
    $self->{dir} = $optArgs->{actualDir};
  } else {
      warn "Cannot create MessageSource of type '$type'\n";
  }
  bless $self, $class;
}
## INSTANCE METHODS
##
# Getters
# MessageSource's type
#
sub type {
  my $self = shift;
  return $self->{type};
}
# MessageSource's fileName
#
sub fileName {
  my $self = shift;
  return $self->{fileName};
}
# render the MessageSource object as XML
#
#sub xmlElem {
  #my $self = shift;
  #my $messageSourceElem = XML::LibXML::Element->new('MessageSource');
  #for my $prop (keys %{$self}) {
    #if ($prop eq 'type') {
      #$messageSourceElem->setAttribute('type', $self->{type});
   # } else {
      #my $subElemType = $prop;
      #my $subElem = XML::LibXML::Element->new($prop);
      #my $messageSourceValueText = XML::LibXML::Text->new($self->{$prop});
      #$messageSourceElem->appendChild($subElem);
      #$subElem->appendChild($messageSourceValueText);
    #}
  #}
  #return $messageSourceElem;
#}
#sub xmlString {
  #my $self = shift;
  #return $self->xmlElem->toString();
#}
1; # end of MessageSource.pm
```

```
# created 9-2-02 by Liz Derr
# modified 11-20-02 by LD -added Finger_length to Finger table
# these tables supercede those in the dbtables folder
# modified 6-26-2003 changed Finger_weight from smallint to decimal,
#        moved user preference db to separate file
# $Id: new_toaster_db_setup.sql,v 1.25 2004/05/26 07:47:50 lizd Exp $
# 2-26-04 removed Finger_name table - not used
# USE accessio_handprints;
##################################################################
#create category and related tables
DROP TABLE IF EXISTS Category;
CREATE TABLE Category (
Category_id  smallint unsigned primary key not null,
Category_name  varchar(20) not null,
Category_desc  varchar(60),
Category_lastmod_date timestamp,
Unique U_Category_name(Category_name)
);
#
#
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("1","Unreviewed",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("2","Adult",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("3","Money Making",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("4","Free Stuff",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("5","Gaming/Casino",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("6","Stuff For Sale",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("7","Virus",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("8","Other",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("9","Not Sure",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("10","Unclassified Spam",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("11","Not Spam",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("12","Loans",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("13","Prescriptions",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("14","Scam",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("15","Asian",NOW());
```

```
##################################################################
#create tables for Fingers
DROP TABLE IF EXISTS Finger;
CREATE TABLE Finger (
Finger_id  int unsigned primary key auto_increment not null,
Finger_Finger_name_id int not null,
Finger_Message_id int not null references Message(Message_id),
Finger_Finger_set_id int not null references Finger_set(Finger_set_id),
Finger_digest_code char(22) not null,
Finger_weight  decimal(6,4) not null default 1,
Finger_length  int unsigned,
Finger_inserted_on datetime,
Finger_updated_on timestamp,
Index   I_Finger_digest(Finger_digest_code),
Index   I_Finger_setid(Finger_Finger_set_id),
Index   I_Finger_msgid(Finger_Message_id)
);
DROP TABLE IF EXISTS Finger_set;
CREATE TABLE Finger_set (
Finger_set_id  int unsigned primary key auto_increment not null,
Finger_set_Message_id int not null references Message(Message_id),
Finger_set_parentset_id int not null references Finger_set(Finger_set_id),
Finger_set_tree_index int,
Finger_set_inserted_on datetime,
Finger_set_updated_on timestamp,
Index I_Finger_set_msgid(Finger_set_Message_id),
Index I_Finger_set_parentsetid(Finger_set_parentset_id)
);

##################################################################
#create tables for Message
DROP TABLE IF EXISTS Message;
CREATE TABLE Message (
Message_id  int unsigned primary key auto_increment not null,
Message_Category_id smallint unsigned references Category(Category_id),
Message_filename char(32) not null,
Message_pathname varchar(64) not null,
Message_inserted_on datetime,
Message_updated_on timestamp,
Unique  U_Message_filename(Message_filename),
Index   I_Message_category(Message_Category_id)
);
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
# $Id: TextParagraph.pm,v 1.4 2004/02/16 03:59:40 lizd Exp $
#
#
package MiaVia::TextParagraph;
use strict;
# take a text line and paragraph collector
# if the line is blank, then return done = true(1)
# else add alpha chars to paragraph and return
# done = false (0)
#
# this is a good candidate for an object, at least
# the paragraph object which includes the string and
# whether or not it's done
sub collect_paragraph {

    my $line = shift;
    my $paragraph = shift;
    my $done=0;

  if ((length($paragraph) > 20) && ($line =~/^\s$/)) {
     # if we've collected more than 20 chars, and
     # line contains only white space, then we're done
     # with this paragraph
     $done = 1;
  } else {
     # extract only alpha characters
     $line =~ s/[^a-zA-Z]//g;
     if ($line ne "") {
        # if the line is not blank now, then
        # add the alpha chars to the collector
        $paragraph = $paragraph.$line;
     }
  } # end else
return ($paragraph, $done);
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                     Copyright 2002,2003
#                     All Rights Reserved
#  $Id: NewMatch.pm,v 1.271 2004/03/27 06:41:38 lizd Exp $
#
# created 09-15-02 by Liz Derr
# 11-20-03 optimizations - removed DISTINCT from selects, flattened mesg_rec to just
#          message id since message_digest wasn't used, got rid of debug prints,
#          removed some old commented out code, removed code related to body fingers
#          and attachments, got rid of fudge factor for small messages, made
#          match message weight the divisor to account for noise fingers now,
#          created find_matches and moved code out of the constructor to the new sub,
#          got rid of now obsolete variables such as $max_score, and got rid of
#          counting the fingers and weight of the test message since we aren't
#          using that information now due to using the weight of the matching
#          message as the divisor for the forthcoming noise fingers, got rid
#          of the get_fingers sub since it wasn't necessary
# 2-15-04 changed to get settings from global $AccessioConfig rather than
# Settings.pm; changed to get is_common_finger from DbUtil instead of Settings.pm
# modified 2-20-04 LD incorporated code by Joe Harris
#    added $userMatch and $cacheScore to new()
#          added logic to short-circuit scoring if the
#          message is cached or if the user has either
#          whitelisted or blacklisted the sender
# 2-24-04 added message id and category to hash cache
#
package MiaVia::NewMatch;
use strict;
use MiaVia::DbUtil;
use MiaVia::Utils; # for config
use MiaVia::SysLogEvent qw(debug_msg);
use DBI;
## CLASS METHODS
#
# Create a Match object, given a Message and db handle.
# a match object has a message, a list of digests that it matches,
# and a match list.  The match list is an array of hashes, each
# element in the array has a message_id, message_digest, digest_list,
# and a score.  The digest list is an array of hashes: each element
# has a digest code and a weight. This is all the info needed to
# calculate the score.
sub new {
  my ($class, $message, $dbh, $userMatch, $cacheScore, $common_dbh) = @_;


  # for backward compatibility
  if (!(defined $userMatch)) {
   $userMatch = 0;
  }
  if (!(defined $cacheScore)) {
```

```
  $cacheScore = 0;
 }
 if (!(defined $common_dbh)) { # common fingers can be in a different db
  $common_dbh = $dbh;
 }

 MiaVia::DbUtil->load_common_fingers_and_sublinks($common_dbh);

 my @matchList = ();
 my $matchCount = 0;
 my $self = { message => $message
           , matchList => \@matchList
           };

 my @digestList = keys %{$message->allDigests()};
 $self->{digestList} = \@digestList;

 # check to see if we need to find matches
 if( ($userMatch > 0) || ($cacheScore > 0) ) {
  my $msg_id;
  my $msg_score;
  my $cat;
if($userMatch == 1) {
 $msg_score = 0; # Whitelisted
 $cacheScore = 0; # overrides cache score
 $msg_id='whitelisted';
 $cat = 1; #unreviewed
 #debug_msg("sender is whitelisted");
} elsif($userMatch == 2) {
 $msg_score = 100; # Blacklisted
 $cacheScore = 0; # overrides cache score
 $msg_id='blacklisted';
 $cat=1; #unreviewed
 #debug_msg("sender is blacklisted");
} elsif($cacheScore > 0) {
 my $md5hash = $message->getMD5();
 my $tmp = MiaVia::Utils->get_mysql_servername();
 my $bdbh = MiaVia::DbUtil::openHashCacheDb('miavia_wrtr',$tmp);
           $msg_score = $cacheScore;
           $msg_id = MiaVia::DbUtil::cacheId($md5hash,$bdbh);
     $cat = MiaVia::DbUtil::cacheCategory($md5hash,$bdbh);
     #debug_msg("message is cached");
}
push @matchList, ({message_id => $msg_id
      ,score => $msg_score
      ,category => $cat
      });
 } else { # generate the match list
  # load the common fingers from the database into a global hash
  # MiaVia::DbUtil::load_common_fingers_and_sublinks();
```

```
   my $matchREF = find_matches($dbh, $message);
   $self->{matchList} = $matchREF; # reference to the match list

  }
  bless $self, $class;
}
## INSTANCE METHODS
##
# get the Match object's fingerprint list
#
sub DigestList {
  my $self = shift;

  return $self->{digestList};
}
# get the list of matching messages
#
sub matchList {
  my $self = shift;
  return $self->{matchList};
}
#
#
sub message {
  my $self = shift;
  return $self->{message};
}
# get the list of matching messages
#
sub bodyScore {
  my $self = shift;
  return $self->{matchList};
}
##########################################################
# Utility SubRoutines
##########################################################
# returns the matchList and the count of matches computed
sub find_matches {
 my $dbh = shift;
 my $message = shift;

        my $orig_length = $message->getFingLengthTotal();

        my %para_hash = %{$message->allDigests()};
 my @digest_list = keys %para_hash;

        my @messageIdList =();
        my @matchList;
```

```
       my $stop_looking_score = MiaVia::Utils->get_stop_looking();
       my $short_thresh = MiaVia::Utils::get_short_message();

       # get list of messages with matching fingers
 # remove common fingers from finger list for query
 my @uncommon_paralist;
  foreach my $digest (@digest_list) {
       # if (!(MiaVia::DbUtil->is_common_finger($dbh,$digest))) {
       if (!(MiaVia::DbUtil->is_cached_common_finger($digest))) {
         push @uncommon_paralist, $digest;
        }
   }
   my %MsgSeen = (); # so only unique messages added to list
       my $queryStr = qq{ SELECT Finger_Message_id
    FROM Finger WHERE Finger_digest_code IN };
my $digest = shift @uncommon_paralist;
if (defined $digest) { # only do this if there is at least one digest
 $queryStr .= "(\"$digest\"";
 foreach $digest (@uncommon_paralist) {
       $queryStr .= ",\"$digest\"";
 }
 $queryStr .= ")";

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();

 while (my ($msgId) = $sth->fetchrow_array()) {
       push (@messageIdList, $msgId) unless $MsgSeen{$msgId}++;
 }
}

# get wild link matching message ids
my $fingerSet = $message->fingerSet(); # finger set has sub links
my @SubLinkElems = @{$fingerSet->allSubLinks()};
#my @sublinkList = ();
       my %uniqueSubLinks = ();
foreach my $subLinkElem (@SubLinkElems) {
 foreach my $sublink (@{$subLinkElem->sublink_list()}) {
  $uniqueSubLinks{$sublink}++;
 }
}

# dedupe sublink list
#foreach my $subLinkString (@sublinkList) {
# $uniqueSubLinks{$subLinkString}++;
 #debug_msg("sublink is $subLinkString");
#}

my @uniqueSubLinkList = keys %uniqueSubLinks;
my %wildValues = %{WildLinkScore($dbh, \@uniqueSubLinkList,
   $fingerSet->getAllLinkValues())};
# add any new message ids discovered in the wildScores to the message ID list
foreach my $msg_id (keys %wildValues) {
      push (@messageIdList, $msg_id) unless $MsgSeen{$msg_id}++;

}

  # now, for each matching message get list of its fingers, with weights
  my $msg_count =0;
   foreach my $msg_rec (@messageIdList) {
 $msg_count++;
 my $queryStr = qq{ SELECT f.Finger_digest_code,
                 f.Finger_weight, f.Finger_Finger_name_id, f.Finger_length
                 FROM Finger f
    WHERE f.Finger_Message_id=};
         $queryStr .= "\"$msg_rec\"";
 my $sth = $dbh->prepare($queryStr);
 $sth->execute();

        my @listOfFingers = ();
 my %seen = (); # so only unique fingers added to list
        my $db_msg_finger_count = 0;
        my $db_msg_weight = 0;
        my $db_msg_length = 0;
 while (my ($digest_code, $weight, $type, $finglen) = $sth->fetchrow_array()) {
  $db_msg_length = $db_msg_length + $finglen;
  #if (!(MiaVia::DbUtil->is_common_finger($dbh,$digest_code))) {
          # if it's not a common finger
          if (!(MiaVia::DbUtil->is_cached_common_finger($digest_code))) {
   if (!(defined $weight) || ($weight == 0)){
     $weight = 1;
    }
    $db_msg_finger_count++;
    $db_msg_weight = $db_msg_weight + ($weight * $finglen);
    # note: @listofFingers is not a list of finger objects,
    # but rather a list of hashes containging db records
    # with unique digest codes
    $seen{$digest_code}++;
    if ($seen{$digest_code} == 1) {
     push @listOfFingers, ({digest => $digest_code, weight =>
     $weight, type => $type, finglen => $finglen})
    }
   } # end if not common
 } # end while



 my $msg_score =0;
              my $msg_divisor = $db_msg_weight; # use weight of possible match message as divisor, so we ignore
noise fingers
```

```
            #debug_msg("message length is $db_msg_length message denominator is $msg_divisor");
foreach my $finger_row (@listOfFingers) {
   # note: @listofFingers is not a list of finger objects,
    # but rather a list of hashes containing db records
   if (exists $para_hash{$finger_row->{digest}}) {
        # add the weight from the db times the length of the finger string to the score
        $msg_score = $msg_score + ($finger_row->{weight}*$finger_row->{finglen});
        my $type = $finger_row->{type};
        # extra weight on HTML fingers
        if (($type eq '4') || ($type eq '5') || ($type eq '6')) {
          # reduced from 8 to 4 on 2-16-04
          # changed back to 8 on 3-16-04
               $msg_score = $msg_score + (8*$finger_row->{finglen});
               $msg_divisor = $msg_divisor + (8*$finger_row->{finglen});
        }
    } # end if exists
 } # end of loop for each finger in the db message
# add value of wild link matches to score and denominator
if ($wildValues{$msg_rec}) {
 #my $printstring = $wildValues{$msg_rec};
 #debug_msg("adding wild value of $printstring to score for $msg_rec");
 $msg_score = $msg_score + $wildValues{$msg_rec};
 $msg_divisor = $msg_divisor + $wildValues{$msg_rec};
}

if ($msg_divisor == 0) { # avoid divide by zero
 $msg_score = 0;
 warn "message divisor is 0\n";
} else {
 $msg_score = ($msg_score/$msg_divisor)*100; # normalize it
}
#my $msg_category = MiaVia::DbUtil::getCategory($dbh, $msg_rec);

# use relativity only for small spam messages


if ($db_msg_length < $short_thresh) {
 # reduce score by ratio of smallest/biggest message length
 my $relativity = 1;
 #debug_msg("orig length is $orig_length db msg length is $db_msg_length");
 if ( $db_msg_length < $orig_length) {
  #$relativity = ($orig_length / $db_msg_length);
 #} else {
  $relativity = ($db_msg_length / $orig_length);
 } # so relativity should be <= 1, and a relative indication of
 # the difference in message lengths
 my $old_score = $msg_score;
 $msg_score = $msg_score * $relativity;
 #if (($old_score > 40) && ($msg_score < 40) ) {
 # print STDERR "UNDER THRESHOLD rel:$relativity, msg_id:$msg_rec,";
```

```
  # print STDERR " old:$old_score, new:$msg_score, ";
  # print STDERR "msg length: $orig_length, spam length: $db_msg_length\n";
  #} else {
  # print STDERR "RELATIVITY USED rel:$relativity, msg_id:$msg_rec,";
  # print STDERR " old:$old_score, new:$msg_score, ";
  # print STDERR "msg length: $orig_length, spam length: $db_msg_length\n";
  #}
 }
 push @matchList, ({message_id => $msg_rec
      ,digest_list => \@listOfFingers
      ,score => $msg_score
      #,category => $msg_category
      });
        if ($msg_score >= $stop_looking_score) { # if we've found a sure match, just stop.
          goto SCORE_DONE;
        }
       } # end foreach
       SCORE_DONE:
       return (\@matchList);

}
# given a reference to an array of sub links and a reference to
# an array of link finger values, get the matching wild links
# from the database, work though list of wild links to determine
# matches with the values of the link fingers.
# if a match is found, add the (weight*length of the wild link) to
# the matchvalue for the message id.
# Returns a reference to a hash of matchvalues indexed
# by message id for messages that a match was found for
sub WildLinkScore {
 my $dbh = shift;
 my $sublinksREF = shift;   # reference to array of sublinks
 my $linkValuesREF = shift; # reference to array of link finger values
 my %matchValues = ();

 #if (scalar(@{$sublinksREF}) <1) {
  # print STDERR "WildLinkScore called with empty sub link list\n";
 #}

 my %wildLinksByString = %{MiaVia::DbUtil::getWildLinks($dbh, $sublinksREF)};

 # make an array of wild card strings
 my @wildStrings = keys %wildLinksByString;

 # loop through each url in the link finger value list
 # check to see if it matches any of the wild link strings
 foreach my $url (@{$linkValuesREF}) {
  foreach my $wstring (@wildStrings) {

   if ($url =~ m<$wstring>) {
     foreach my $elem (@{$wildLinksByString{$wstring}}) {
      #debug_msg("match found for wild string $wstring");
      if ($matchValues{$elem->{msgid}}) {
       $matchValues{$elem->{msgid}} =
        $matchValues{$elem->{msgid}}
        + ( $elem->{weight} * length($wstring) );
      } else {
        $matchValues{$elem->{msgid}} =
        $elem->{weight} * length($wstring);
      }
     } # end for each element in the wild link hash for string
    } # end if match
   } # end foreach wild string
  } # end for each link value (url)
  return (\%matchValues);
 }
 1; # end of NewMatch.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: NewFingerSet.pm,v 1.41 2004/06/04 02:30:53 lizd Exp $
#
# modified 9-24-02 by Liz Derr to support multiple fingers, this now obsoletes
#                 FingerSet.pm
# modified 12-03-03 by LD added sublinks to FingerSets.  Sublinks are for look
#                 up purposes, and are not used for scoring like fingers are
# 2-15-04 clear of all Settings references
# 6-3-04 removed libXML

package MiaVia::NewFingerSet;
## A FingerSet is a collection of Finger objects, all originally
## extracted from the same MIME entity in an email message. A
## FingerSet may contain other FingerSets, in the case when the
## entity contains sub-entities.
use strict;
#use XML::LibXML;
use MiaVia::NewFinger;
use MiaVia::SubLink;
## CLASS METHODS
##
# Create a new FingerSet, given a list of Fingers
#
sub new {
  my ($class,              # FingerSet or sub-class
     @subElemList) = @_;  # Finger and FingerSet elements
  my $fingers = [];
  my $fingerSets = [];
  my $subLinks = [];
  my $self = {  fingers => $fingers
           ,  fingerSets => $fingerSets
           ,  subLinks => $subLinks
           };

  foreach my $subElem (@subElemList) {
    if (ref($subElem) eq "MiaVia::NewFinger") {
      push @{$fingers}, $subElem;
    } elsif (ref($subElem) eq "MiaVia::NewFingerSet") {
      push @{$fingerSets}, $subElem;
    } elsif (ref($subElem) eq "MiaVia::SubLink") {
      push @{$subLinks}, $subElem;
    }
  }
  bless $self, $class;
}
## INSTANCE METHODS
##
```

```
# get sub-FingerSets
#
sub fingerSets {
  my $self = shift;
  return $self->{fingerSets};
}
# get Fingers
#
sub fingers {
  my $self = shift;
  return $self->{fingers};
}
# returns a list of SubLink objects
sub subLinks {
 my $self = shift;
  return $self->{subLinks};
}# returns a list of SubLink objects for this fingerset
sub allSubLinks { # returns a list of sublink object for this fingerSet
    # and all child fingersets
 my $self = shift;

 my @allSubLinkList = ();
  # put my sublinks into the list
  foreach my $subElem( @{$self->{subLinks}}) {
  push @allSubLinkList, $subElem;
  }

  #recursively put my children's sublinks into the list
  foreach my $fingerSet (@{$self->{fingerSets}}) {
 push @allSubLinkList, @{$fingerSet->allSubLinks()};
  }
  return \@allSubLinkList;
}
sub allFingerLengths { # returns the sum of all the fingers in this finger set
    # and all child fingersets
 my $self = shift;

 my $total_length =0;
  foreach my $finger( @{$self->{fingers}}) {
  $total_length = $total_length + $finger->finglen();
  }

  #recursively put my children's sublinks into the list
  foreach my $fingerSet (@{$self->{fingerSets}}) {
 $total_length = $total_length + $fingerSet->allFingerLengths();
  }
  return($total_length);
}
# get a hashset of all digests
# each element of the hash is a digest and the weight of the digest
# (length of the original string)
#
sub allDigests {
  my $self = shift;
  my %allDig = ();
  my $print_string;
  my $print_string2;


  foreach my $finger (@{$self->{fingers}}) {
    if (defined $finger->value()) {
      my $len = length($finger->value());
      $allDig{$finger->digest()} = $len;
    }
  }
  foreach my $fingerSet (@{$self->{fingerSets}}) {
  # for each sub fingerset

     my %hash = %{allDigests($fingerSet)};
     foreach my $key (keys %hash) {
     # add all digests from it's fingers/sub-finger sets

         $allDig{$key} = $hash{$key};
     }
  }
  return \%allDig;
}
# returns a reference to an array of finger values for all link type
# fingers in the finger set (type_id = 4,5,or 6)
sub getAllLinkValues {
 my $self = shift;
 my @linkValues = ();

 # get values for my fingers
 foreach my $finger (@{$self->{fingers}}) {
  my $ftype = $finger->type_id();
      if (($ftype eq '4') || ($ftype eq '5') || ($ftype eq '6') ) {
       push @linkValues, $finger->value();
      }
     }

     # get values for my fingersets (recursive)
     foreach my $fingerSet (@{$self->{fingerSets}}) {
       my @moreValues = @{getAllLinkValues($fingerSet)};
       push @linkValues, @moreValues;
      }

 return (\@linkValues);
}
```

```
# redigest all fingers
#
sub redigest {
  my $self = shift;
  foreach my $finger (@{$self->{fingers}}) {
    $finger->redigest();
  }
  foreach my $fingerSet (@{$self->{fingerSets}}) {
    $fingerSet->redigest();
  }
}
# create an XML element representing the Finger
#
#sub xmlElem {
  #my $self = shift;
  #my $fingerSetElem = XML::LibXML::Element->new('FingerSet');
  #for my $finger (@{$self->{fingers}}) {
    #my $fingerElem = $finger->xmlElem;
    #$fingerSetElem->appendChild($fingerElem);
  #}
  #for my $subFingerSet (@{$self->{fingerSets}}) {
    #my $subFingerSetElem = $subFingerSet->xmlElem;
    #$fingerSetElem->appendChild($subFingerSetElem);
  #}
 # return $fingerSetElem;
#}
# render the FingerSet as an XML string
#
#sub xmlString {
  #my $self = shift;
  #return $self->xmlElem->toString();
#}
1; # end of FingerSet.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: PipelineRegisterSpam.pm,v 1.2 2004/03/18 08:55:13 lizd Exp $
#
#
# this is a derivation of RegisterSpam.pm
# it now checks for a duplicate in the db before
# registering the spam
#
# created 7-24-02 to use LogEvent logging instead of Log
#                 added error handling
# modified 8-13-02 by Liz Derr to add removal of forwards
# modified 9-03-02 by Liz Derr to add call for help routine
# modified 11-09-02 by Liz Derr to call Queue Manager for CC
# modified 11-11-02 by LD to write file to dupes first, and then
# rename to nominations if no mathc is found
# modified 3-19-03 more logging to debug apparent "leaks"
# 3-21-03 added moving queue dupes to the queue dupe dir#
#############PIPELINE###############
package EventRegisterSpam;
use strict;
use MiaVia::WWW::QueueManager;
use Digest::MD5 qw( md5_hex );
use MIME::Parser;
use MiaVia::PipelineSettings;
use MiaVia::PipelineLogEvent;
use MiaVia::Message;
use MiaVia::DbUtil;
use MiaVia::AccessioWrapper;
use MiaVia::Utils;
use MiaVia::PipelineUtils;
# given a MIME::Entity, compute the hex MD5 digest,
# and write them to a file (named after the digest) in the
# specified directory. Check to see if the file is already
# in the database.  If it is, move the file to the dupe dir
# as specified in Settings.pm. Log the header info of the
# message using LogEvent.
#
sub eventRegisterSpam {
  my ($entity, $outputDir) = @_;
  my $hexDig = md5_hex($entity->stringify);
  my $logDir = MiaVia::Settings->dirs('log');
  my $tmpDir = MiaVia::Settings->dirs('mime_temp');
#  my $dupeDir = MiaVia::Settings->dirs("raw_dupes");
#  my $bkupDir = MiaVia::Settings->dirs("forward_backups");
   my $dupeDir = "/usr/local/accessio/data/raw_dupes/";
  my $bkupDir = "/usr/local/accessio/data/forward_backups/";
  my $dbh = MiaVia::DbUtil::openDb('miavia_rdr');
  my $fileName = "$dupeDir$hexDig.txt";
  my $logString;
  my $matchID;
  my $filter;
  my @scrubbed_msg;
  my $line;
  my $head = $entity->head;
  my $From = $head->get("From");
  my $Subject = $head->get("Subject");

# here is where we look for forwards in the message
# body, and remove them if present
my $remove_needed = MiaVia::PipelineUtils::look_for_forwards($entity);
if ($remove_needed eq 'Y') {
   my @newbody = MiaVia::PipelineUtils::remove_forwards($entity->stringify_body, $From);
   # make a backup copy of the original file
   open (OUT, ">$bkupDir$hexDig.txt")
     or warn("EventRegisterSpam: can't open file $bkupDir$hexDig.txt: $!");
   print (OUT $entity->stringify);
   close OUT;
   chmod 0755, "$bkupDir$hexDig.txt";
   # change the message
   push @scrubbed_msg, $entity->stringify_header; #put header first
   push @scrubbed_msg, @newbody; #put body after header
   # note in the log that forwards were removed
   $logString = "NOQ FORWARDS REMOVED "
} else {
 push @scrubbed_msg, $entity->stringify;
} # end if remove needed

# write out the message file, with the hex digest file name, into
# the output directory

  open (OUT, ">$fileName") or warn("EventRegisterSpam: can't open file $fileName: $!");
  foreach my $line (@scrubbed_msg) {
    print (OUT $line);
  }
  close OUT;
  chmod 0755, "$fileName";
# look for a match
  my $spamScore;
  ($matchID, $filter, $spamScore) = MiaVia::AccessioWrapper::findMatch
              ($hexDig, $dupeDir, "txt",'40');

  if ($spamScore >= 0) { #not an empty message
# if match exists, move the file to the dupes directory
   if ($matchID) {
       DUPE:
       $logString = $logString."NOQDUPLICATE";
   } else {
       rename $fileName,"$outputDir$hexDig.txt" or
```

```
         warn ("can't rename file $fileName to $dupeDir$hexDig.txt: $!");
        $fileName = "$outputDir$hexDig.txt";
      $logString = $logString."NOQ - NOT IN DB YET";
    }
   } else { # empty message
    unlink $fileName;
    $logString = $logString."NOQ - EMPTY MESSAGE UNLINKED";
   }
  #print "eventRegisterSpam: writing message to file $fileName\n";
# extract header details for the log using MIME parser
  my $MessageId = $head->get("Message-ID");
  my $Sender = $head->get("Sender");
  my $Date = $head->get("Date");
  my $To = $head->get("To");

 # get the most recent db version number
  my $db_version = MiaVia::Utils::get_dbversion($dbh);

# Setup the logging
  my $klogger = new LogEvent(
      TYPE => 'register'
    , LOG_DIR => $logDir
    , FILTER_VERSION => $filter
    , DB_VERSION => $db_version
   , PATH_TO_CRONOLOG => MiaVia::Settings->cronopath()
  );
# now finally log the whole thing
  $klogger->logCCEvent(
         STATUS => "NOQREGISTERED"
        , COMMENT => $logString
        , SENDER => $Sender
        , FROM => $From
        , TO => $To
        , SUBJECT => $Subject
        , MAIL_MSG_ID => $MessageId
        , DB_MSG_ID_LIST => $matchID
        , FILENAME => $fileName
        , MAIL_DATE => $Date
        );
}
###########################################################
# for the new command center, check incoming messages against
# both the spam db and the queued messages db. Insert if brand
# new; queue message appropriately
# 11-10-02 by Liz Derr
###########################################################
sub queueSpam {
  my ($entity) = @_;

  my $outputDir = "/usr/local/accessio/data/CC_nominations/";
  my $hexDig = md5_hex($entity->stringify);
#  my $fileName = "$outputDir$hexDig.txt";
  my $logDir = MiaVia::Settings->dirs('log');
  my $tmpDir = MiaVia::Settings->dirs('mime_temp');
#  my $dupeDir = MiaVia::Settings->dirs("raw_dupes");
#  my $queuedupeDir = MiaVia::Settings->dirs("queue_dupes");
#  my $bkupDir = MiaVia::Settings->dirs("forward_backups");
  my $dupeDir = "/usr/local/accessio/data/raw_dupes/";
  my $queuedupeDir = "/usr/local/accessio/data/queue_dupes/";
  my $bkupDir = "/usr/local/accessio/data/forward_backups/";
  my $dbh = MiaVia::DbUtil::openDb('miavia_rdr');
   my $fileName = "$dupeDir$hexDig.txt";
  my $logString;
  my $matchID;
  my $filter;
  my @scrubbed_msg;
  my $line;
  my $score = 0;
  my $result =0;
  my $message; # used for inserting the new queued message
  my $message_filename;
 # get the most recent db version number
 # my $db_version = MiaVia::Utils::get_dbversion($dbh); 2-15-04 NOT USED

#########################################################
# clean up the message - remove forwards, check for dupe
  my $head = $entity->head;
  my $From = $head->get("From");
  my $Subject = $head->get("Subject");
  # comented out 3-19-03
  #if (defined($Subject)) {
  #   if ($Subject =~ /Spam\((4|5|6|7|8|9|10)/) {
  #    goto DUPE;
  #   }
  #}
# here is where we look for forwards in the message
# body, and remove them if present
my $remove_needed = MiaVia::Utils::look_for_forwards($entity);
if ($remove_needed eq 'Y') {
   my @newbody = MiaVia::Utils::remove_forwards($entity->stringify_body, $From);
   # make a backup copy of the original file
   open (OUT, ">$bkupDir$hexDig.txt");
#      or MiaVia::Utils::call_for_help("register spam problem", "EventRegisterSpam: can't open file $bkupDir$hexDig.txt:
$!");
   print (OUT $entity->stringify);
   close OUT;
   chmod 0755, "$bkupDir$hexDig.txt";
   # change the message
   push @scrubbed_msg, $entity->stringify_header; #put header first
   push @scrubbed_msg, @newbody; #put body after header
   # note in the log that forwards were removed
   $logString = "Q FORWARDS REMOVED "
} else {
 push @scrubbed_msg, $entity->stringify;
} # end if remove needed
############################################################
# write out the message file, with the hex digest file name, into
# the output directory
  open (OUT, ">$fileName") or
   MiaVia::Utils::call_for_help("Queue spam can't write file","EventRegisterSpam: can't open file $fileName: $!");
  foreach my $line (@scrubbed_msg) {
    print (OUT $line);
  }
  close OUT;
  chmod 0755, "$fileName";

############################################################
# look for a match in the spam database
  ($matchID, $filter, $score) = MiaVia::AccessioWrapper::findMatch
               ($hexDig, $dupeDir, "txt",'40');
# if match exists, move the file to the dupes directory
  if ($score >= 0) { #not an empty message
   if ($matchID) {
        rename $fileName,"$dupeDir$hexDig.txt"; #or
 #          #MiaVia::Utils::call_for_help("register spam problem","can't rename file $fileName to $dupeDir$hexDig.txt: $!");
        $fileName = "$dupeDir$hexDig.txt";
        DUPE: # the file was never written if we jump to this label
        # print "This message is a duplicate of $matchID\n";
        $logString = $logString."HANDPRINT DUPLICATE of $matchID scored $score ";
   } else {
     ############################################################
     # look for a match in the queued_messages db, which has both
     # unreviewed messages, and non-spam messages
     ($matchID, $message_filename, $score, $message) = MiaVia::AccessioWrapper::findQueuedMatch
               ($hexDig, $dupeDir, "txt",'40');
     if ($matchID) {
      # we found a dupe in the queued_messages db, move it to queue dupe dir

      rename $fileName,"$queuedupeDir$hexDig.txt";
      $fileName = "$queuedupeDir$hexDig.txt"; # for logging purposes
      # send the matched message to the command center queue for dupe counting
      # commented out until we get the queue daemon working smoothly
 #      $result = MiaVia::WWW::QueueManager->insertMessage($message_filename,time());
      #if ($result > 0) { commented out 3-19-03
      #    $logString = $logString."QDB DUPE of $matchID score: $score SENT TO QUEUE";
      #} else {
         $logString = $logString."QDB DUPE of $matchID score: $score $fileName ";
      #}
      } else {
  # a message we've never seen before, so insert it
  # into the queued_messsages db
  rename $fileName,"$outputDir$hexDig.txt"; #or
         #MiaVia::Utils::call_for_help("register spam problem","can't rename file $fileName to $dupeDir$hexDig.txt: $!");
       $fileName = "$outputDir$hexDig.txt";
  $dbh = MiaVia::DbUtil::openQueueDb('miavia_wrtr');
  my $msgDest = new MiaVia::MessageDestination('database',{ dbh => $dbh});
  my $messageID = MiaVia::NewMessageDelivery::deliverMessage($message, $msgDest);

  # send the message to the command center queue
  # commented out until we get the queue daemon working smoothly
 # $result = MiaVia::WWW::QueueManager->insertMessage($hexDig,time());
  #if ($result > 0) { commenetd out 3-19-03
     #    $logString = $logString."Q NEW MESSAGE $messageID score: $score SENT TO QUEUE";
     #} else {
        $logString = $logString."Q NEW MESSAGE $messageID score: $score $fileName ";
     #}
     }
   }
  } else { # empty message
   unlink $fileName;
   $logString = $logString."Q EMPTY MESSAGE UNLINKED";
  }
##########################################################
# log the event
# extract header details for the log using MIME parser
  my $MessageId = $head->get("Message-ID"); # this is the email message id
  my $Sender = $head->get("Sender");
  my $Date = $head->get("Date");
  my $To = $head->get("To");
# Setup the logging
  my $klogger = new LogEvent(
     TYPE => 'register'
    , LOG_DIR => $logDir
    , FILTER_VERSION => $filter
    , DB_VERSION => $db_version
   , PATH_TO_CRONOLOG => MiaVia::Settings->cronopath()
  );
# now finally log the whole thing
  $klogger->logCCEvent(
         STATUS => "REGISTERQ"
       , COMMENT => $logString
       , SENDER => $Sender
       , FROM => $From
       , TO => $To
       , SUBJECT => $Subject
       , MAIL_MSG_ID => $MessageId # this is the email message id
       , DB_MSG_ID_LIST => $matchID
       , FILENAME => $fileName
       , MAIL_DATE => $Date
      );
```

```
############################################################
# return the result of the queue operation
return ($result);
}
1; # EventRegisterSpam.pm
```

```
#!/usr/local/bin/perl
use strict;
my $count = 0;
my ($indir, $outdir) = @ARGV;
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
if (!($outdir =~/\/$/)) {
 $outdir = $outdir.'/';
}
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    while (my $line = <IN>) {
        if (

($line =~ /\&winner\&_m01/) ||
# actual search string for above is "&winner&_m01"

#        add more conditional lines below this line
#####################################################
($line =~ /shogu\/hotel/) ||
 # actual search string for above is "shogu/hotel"

($line =~ /\/pher\/o.html/) ||
 # actual search string for above is "/pher/o.html"

($line =~ /us\/alpha\/\?hpsales/) ||
 # actual search string for above is "us/alpha/?hpsales"

($line =~ /us\/hgh\/\?hpsales/) ||
 # actual search string for above is "us/hgh/?hpsales"

($line =~ /instrhh.com/) ||
 # actual search string for above is "instrhh.com"
# To add new strings to look for,
# copy this line below, and place the new line somewhere between the first and last
# conditional lines (they are special), then, add your string, without any
# punctuation characters, between the two /s, and remove the leading "#" to
# uncomment the line, and you are good to go
#
# if you need to use the following chars in your string, preceed them with
# a backslash: # / @ ? & ! = So, a string like "gal@yahoo.com" would need to become
#  "gal\@yahoo.com"
#
# below are some blank conditional lines, commented out.
# ($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#($line =~ //) ||
#
################################################
# add more conditional lines above this line
($line =~ /\/\?AFF_ID\=/)
# actual string is "/?AFF_ID="
) { # we found an uncatchable dupe
         $count++;
         rename  "$indir$fname","$outdir$fname"
          or die ("can't rename file $indir$fname to $outdir$fname: $!");
  goto DONE;
        }  # end if
     } # end while still lines in file
     DONE:
     close IN;
} # end foreach
#print "$count files moved\n";
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("8A4dwPetCu18+7sNX1ebBQ" , "clickherenow"  ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("vDlw1j0KysQmtgvEsExwzQ" , "contenttypetexthtml"   ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("6QrcYYVv0QTM2YZvgKdRbw" , "contenttypetextplain" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("EQ4ESvkhu8kWt4QrBJXZ2g" , "contenttypetexthtmlinline" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("oBNljjA4f3pgza8KzyaFDQ" , "unsubscribe" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1B2M2Y8AsgTpgAmY7PhCfg" , ""  ,NOW());   # the blank finger

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("QZ6sBCnqv3u1DyD5P1vEqw" , "textplain"  ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("G7q7Pu2fjFYk9RnY20keZA" , "contenttransferencodingbit"  ,NOW());

#INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
# VALUES("" , ""  ,NOW());  # the null digest code

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("4WyZTgfcV5h1bikUWjxKKQ" , "ifsupportemptyparas" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("lrObi5XgFsedEE2DOVuBMw" ,
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("qFh1pRNmY9QyM3Byv3Sbqg" , "trtabletdtr" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("zH7Wac+l8gHDKXxqkeHRjQ" ,
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("u5nF+loraYHwzVX1Cch4Ww" , "tounsubscribeemailtodebiancdrequestlistsdebianorg" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ksfWzZJSDwu55proDUU20Q" , "withasubjectofunsubscribetroublecontactlistmasterlistsdebianorg"
,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("SfaKXIST7CwL9ImCHCH8Ow" , "hi" ,NOW());
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("GJbegN6xd4gGmBt7JYwUAQ" , "tounsubscribeemailtodebianpilotrequestlistsdebianorg" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("fqcH+9/Zd/l0aUMcUUELbA" , "doyouyahoothenewyahoosearchfastereasierbingo" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("cdPotCeSteR2gE9Pf73cWA" , "thanks" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("bEiHSRdXkKZfyIFGoF/eWg" , "charsetisocontenttransferencodingquotedprintable" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("/koJ9huIAdFfbkl9E57gBQ" , "thisisamultipartmessageinmimeformat" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1U1EtxdGGTb1vCgxiuExBA" , "contenttypemultipartalternative" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("k7J76e17tFejPcOBHUuZFQ" , "divaligncenter" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("4pBuggsUVfjYm+2i9zpvSg" , "tablecellspacingcellpaddingwidthborder" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("xpHo7FUeS5smmPn18wLxJA" , "widthtdtrtr" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("4u6idBdKOTjgD+1mX29WBg" , "divalignleft" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("FM71Ax5AJnLE6GJt3CG9gA" , "contenttransferencodingbase" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("XUfKZwnUPR+IXVj34NEbfA" , "mimeversion" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("GXTdtjk01VXxbleNjMaJIA" , "contenttypetexthtmlcharsetiso" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("NjnohZ6hpBgE8RjYoEl28Q" , "tdtrtablebody" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Vefk3qhS1LUVDKFkghECpw" , "trtdcolspan" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("b//xR/iqzjlBNV/pc+Slrw" , "http://us.rd.yahoo.com/search/mailsig/*http://search.yahoo.com" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
```

```
 VALUES("Dq4AQh1C7uV80ZqUWwA/ww" , "doyouyahoothenewyahoosearch" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("RtOHOsqTQJU+HITI1TNHZw" , "originalmessage" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("CQuXT0471OwecqadzUQcyQ" , "noteforwardedmessageattached" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("742aV7wxqgoa9BzWBrFr6g" , "nontextportionsofthismessagehavebeenremoved" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("E2qHmaeS6PP075AO4Ux/yQ" , "tounsubscribefromthisgroupsendanemailto" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Ym7RGfRTEwU8ErEt2J6lmQ" , "http://www.excite.com" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("mBwXxe91n8QIFLsg9gPV0Q" , "joinexcitehttpwwwexcitecom" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("MbNWOFIQ4wLu8swkOg3nsA" , "themostpersonalizedportalontheweb" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("wXVlLpvA8YLVRYluBvpyhA" , "msnhelpseliminateemailvirusesgetmonthsfree" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("8+FycJXCZB7r4uY+st4jvg" , "http://join.msn.com/" ,NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Av30ivFn5laSkYgcWz8qig" , "tdclassxltd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("XjEjMElYmj1KstzHTwAsIQ" , "tdalignmiddlewidthfontfaceverdana",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("APDAhfcrKLfvPNN3zgX+vg" , "sizecontfonttd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("+8Aj27Mwo5j6HPqxPPsR2w" , "scripttrtdfontfaceverdanasizea",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("w/jcFZz5kJiIsMYqQGbb0Q" , "msofontcharset",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Bb44/QnUZSgCPMldqD2fzw" , "textdecorationnone",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("PuD5vr7aSBYGkHVhy9x5wA" , "whitespacenowrap",NOW());
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ukoXkgWUZzAaX7Zm7ljtbA" , "verticalalignbottom",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("uk3eEphFWSBu9a/id4J47g" , "msopatternauto",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("T9IwTxceojBwj0RSq9OXMg" , "xlpaddingpx",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("GJKftsehvsFc8vC8gmgozA" , "msobackgroundsourceauto",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1vUN4kH+zKMaPoIFFhPM9A" , "msoignorepadding",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("03GYa9a7FxWLRNKRkMIKoA" , "yearsfonttd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Kvd5i9lqsMbBprfggYUreA" , "fontstylenormal",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("jEBeEBGuAdooLWCqhMxgcw" , "tdalignleftwidthfontfaceverdanasize",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("3H0pALycjvF8oIfvwc2jnQ" , "colorwindowtext",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("eobFI8v2ZX+CJtCPLTkemQ" , "fontsizeptfontweight",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("e6PJTnex3Q4rfBvI/qvHpg" , "fontfamilymonospace",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("4pBuggsUVfjYm+2i9zpvSg" , "tablecellspacingcellpaddingwidthborder",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("gpxEZVd9ssVnXja8a8FjFw" , "doctypehtmlpublicwcdtdhtmltransitionalen",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1ezvJANo77zFP2TqrhbC+g" , "metahttpequivcontenttypecontenttexthtmlcharsetiso",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("sTjKLhKtgIZw7rYPc9F3Ag" , "msonumberformatgeneral",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1Wthsa86LDWMVyyXkchbEA" , "pnbspppnbspp",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("TgbSp5m/MQVHXOhQVlsECg" , "tdheightclassxlstyleheightpttd",NOW());
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("UXkovYiruwfc81z4HaSjkA" , "borderbottomptsolidwindowtext",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("pWzi7cuuMA4i11LFAzmEgw" , "fonttdtrscriptlanguagejavascripttypetextjavascript",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("qFh1pRNmY9QyM3Byv3Sbqg" , "trtabletdtr",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("aVCwN9RqJ53MtDd087uO6g" , "tdalignmiddlewidthfontfaceverdanasizeopenfonttd ",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("5tW4K9fub3KGFjj8L6Bfmw" , "tdcolspanclassxlstylemsoignorecolspantd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("YFbnR4IzIENSFqvEKEhmBw" , "borderrightptsolidwindowtext",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("2MPmcfBeyI2CDZTQx6Ga3Q" , "borderleftptsolidwindowtext",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("amjFkDxDE/1+fUvC/pyRIA" , "textaligncenter",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("xpHo7FUeS5smmPn18wLxJA" , "widthtdtrtr",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("PFDWLPnv5LLP34diDo0KEg" , "scriptlanguagejavascripttypetextjavascript",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("qG0i8t+vquZaj1w4O1Ua8Q" , "trtrtdcolspan",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("/gNqVhcQuzH5pO4FmmYeBw" , "trtrclassxlheightstyleheightpt",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("oXSi+EDNuFxOZqyD9hoC/A" , "brbrbrbrbrbr",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("R2kIDS0E/TaDAGa4qdz2Gw" , "bordertopptsolidwindowtext",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("5TYHPwFzA4XE8oqxifPVDg" , "pclassmsonormal",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("h2iK8ubBYpy3F9vrZNAZFQ" , "tablewidthbordercellspacingcellpadding",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
```

```
 VALUES("pGBt/nH/jf52JAVqBhCJqw" , "tdalignmiddlewidthfontfaceverdanasizefonttd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("p2rpgvcIdT01W66kRZ0k2g" , "metacontentmshtmlnamegeneratorhead",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("kce1l9do1uKNwa6Y6heB9Q" , "msopaginationwidoworphan",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("444OP6+QW6v0QPTb4qmHrQ" , "bordertopnone",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("e55DWs3Xxd66BbAYILxMKQ" , "trvaligntop",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Dl984HSPKmF8BsuDlaFvOA" , "textaligngeneral",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("KAvFnyunSBR0KVhNlhKy3Q" , "metacontentmshtmlnamegenerator",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ws+lAUe/BS5FkWIO1t0H3Q" , "paligncenternbspp",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("dEI/tSA7oXyYMwTSwJlkbg" , "tablewidthbordercellpaddingcellspacing",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("VPr6NTKHr7RHFgnffNzvAg" , "msonumberformat",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("e2PPTqkW8DQC0EpMM8lUVA" , "bodybgcolorffffff",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("U1OASwovKbGIDXjxp4xMtw" , "bordertbody",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("pM/dbOxK1utYhJRAphXVLw" , "untitleddocument",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("d3amTc3lJhbKtuSQB/179g" , "tdtrtabletd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Vefk3qhS1LUVDKFkghECpw" , "trtdcolspan",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("HUwSgKK3c0LD+RxEuCMDzw" , "htmlheadmetahttpequivcontenttypecontenttexthtmlcharsetiso",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("BdC+qlQkT2zosGuVZ1Kj9g" , "stylestylehead",NOW());
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("rTgjHaFq9f3QMPUMJ5WquQ" , "paligncente",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("b832YyrE3fCp1nPs865eSw" , "widthborderatd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("P7vR+oD+bbSS0a3NW4YMVQ" , "borderatdtr",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("0XlOBhOpQTR0s5u0Gmo8AA" , "bestregards",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("13C3nj9hUtTdBe/ROtAwyA" , "msofareastfontfamilytimesnewroman",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("xiq1jWGMJdqgTPH9Wwh/Hg" , "fontsizeptfontfamilytimesnewroman",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("lHwMwHk4oxOOfOtAf3lszA" , "fonttdtrtable",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("UD0Qy+Ii5E77zG+Lm1rrew" , "tablebordercellpaddingcellspacingwidth",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("lPn3nYUr489WxMQA4mUHsw" , "trtablecenter",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("KphOxMPgOtwhay2xHp+QpQ" , "tdalignleftfontfaceverdanasizeyearsfonttd",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("uBmjdzvOb77W3mGRiNBAig" , "trtrclassxlheightstylemsoheightsourceusersetheightpt",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("1vTgdbupkoHsjluTXg70Bw" , "tounsubscribefromthesemailingspleaseclickhere",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("oUQqGTepaLS3moUn2U6DqA" , "fontfamilymsogenericfontfamilyauto",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("nEo8Zwn4nNYL0CFoTTXaeg" , "functionmmopenbrwindowtheurlwinnamefeaturesv",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ANadwZy3usMRPpZHBPDPlg" , "windowopentheurlwinnamefeatures",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("RtOHOsqTQJU+HlTl1TNHZw" , "originalmessage",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("spTMZ2B3zVRPWzK+1fozQQ" , "checkedbyavgantivirussystemhttpwwwgrisoftcom",NOW());
```

```
INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("cvUnVYDq5xjz0X8Y6m5Z/A" , "outgoingmailiscertifiedvirusfree",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("lOZ7gXcDSDwqIlXDuSdb8A" , "versionvirusdatabasereleasedate",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("538yd+xpzoCWB6q1lsg4jQ" , "wwwpaypalcom",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("7uqSaVb3yJ2E5CPYxxTVZw" , "clickhereformcafeecomvirusscanonline",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("nRUsaWMhiAdaIab5aIayBw" , "http://www.paypal.com/",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("cF7IHIHSPQEiMQ+Dgb3yVA" , "http://cgi.ebay.com/ws/ebayisapi.dll",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("m4CvHNW+t5rrg+A0qBhMeQ" , "http://www.ebay.com",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("491fLezC9BbhwkLWqZkzMQ" , "visitebaytheworldsonlinemarketplacetmat",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("7WRqMzTKiR/TRn2xMTchQA" , "http://www.google.com",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("PxIGHJVpW0CjM0jPNAU9+A" , "doyouyahooyahootaxcenterfileonlinecalculatorsformsandmore",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("dqnf7fHePMVNiLYAqvs8pg" , "youcanaccessthisfileattheurl",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("LfEydWBRS63IPmGPQCeNEA" , "tolearnmoreaboutfilesharingforyourgrouppleasevisit",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("xsvhA2LR0t80XEXGIRb++Q" , "http://help.yahoo.com/help/us/groups/files",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("48YqDl1Ev+k0LpyZ/wfWPg" , "httphelpyahoocomhelpusgroupsfiles",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("4H5DgSVlnQwElxMyDMy14w" , "http://tax.yahoo.com",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("iTiHL7+QSMVhJSMvyqcYBQ" , "http://platinum.yahoo.com",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
```

```
 VALUES("MJR5/veruDElclP1oH4Y5g" , "httpswwwpaypalcomprefsnoti",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("PTtA05fHM86L0Mwx/kHyyA" ,
"copyrightpaypalincallrightsreserveddesignatedtrademarksandbrandsarethepropertyoftheirrespectiveowners",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("rQDtJpOOcfSZEV/EZI/x0Q" , "https://www.paypal.com/",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("qm5V80wiVuZ+UHRhFyrNOA" , "https://www.paypal.com/prefs-noti",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("qCfjjuW3tlG16uo2viv/lw" , "http://www.paypal.com/images/paypal_logo.gif",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("Y9VTcaZndslByclx6c0dRw" , "http://www.paypal.com/images/pixel.gif",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("XF3avdBO3Z7Q+Y7xpO5Rlw" , "http://www.paypal.com/images/dot_row_long.gif",NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("G917PwL+k5AfSf/ck13eaw" , "incredimailemailhasfinallyevolvedclickhere",NOW());

 INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("XVZg0En5bfVdx/kaEUZ1AA", "clickinghere", NOW());

  INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ISFEubcL87jGXNFueHxmcQ", "moneybackguarantee", NOW());
  INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("UKkmNgVf3lvcExkpK+AZtw", "clickherethankyou", NOW());

 INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("9uzfC15XAnQys0s1t7w7qg", "tounsubscribe", NOW());

  INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("ydWwKsbblpZzsac/BPA8xA", "privacypolicy", NOW());

   INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("W9b3XhFRK3OQp6dx+1lerw", "tounsubscribegoto", NOW());

   INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("BH5oS/j3TWdpZbxWq6xJsw", "clickheretounsubscribe", NOW());

   INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("0XlOBhOpQTR0s5u0Gmo8AA", "bestregards", NOW());

    INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
 VALUES("6ewJHBYf0fSK87yiJ0qILA", "freeshipping", NOW());
```

```
   INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("m04LvVIS1gJf51XySvFfgA", "allrightsreserved", NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("uwkgGQiSyovIr6yq8L3WVQ", "yoursfaithfully", NOW());

 INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("yh81HbxevdVXj1w/q4cQ5A", "metahttpequivcontenttypecontenttexthtmlcharsetwindows", NOW());

 INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("VOBcsWMMCXm/6leMIGnPLA", "yourssincerely", NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("+XDPFO/V4hZyn1MOOo39WQ", "tounsubscribeclick", NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("+42YvhJI3Yi6xSLhshghQA", "congratulations", NOW());

INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value, CommonFinger_inserted_on)
VALUES("Kz/cpqAkj0iT1T+8Ez+97Q", "belowistheresultofyourfeedbackformitwassubmittedby", NOW());
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
# $Id: DbUtil.pm,v 1.80 2004/06/10 07:33:52 lizd Exp $
#
# modified 11-21-02 by LD to add default setting for MV_DB
#
# 08-17-03 commented out ENV MV_DB use
#         connect string - db:host from config file
#
# 11-21-03 added support for networked databases, support for wildcard links,
#         and HashCache from Joe Harris
# 2-15-04  removed reference to MiaVia::Settings
#         moved dbPassword from Settings to here
# 2-19-04 added getCatString and getCategory
# 2-26-04 added markFingerAsIrrelevant
# 3-25-04 reorganized, removed pipeline specific routines
package MiaVia::DbUtil;
use strict;
use DBI;
our %dbPasswd = (
                 "miavia_rdr" => "XXXXX"
               , "miavia_wrtr" => "XXXXX"
               );


############ Common FInger/SubLink Subs ############
#
our %CommonFingers = ();
our %CommonSubLinks = ();
sub is_cached_common_finger {
  my ($class,$digest) = @_;
  return $CommonFingers{$digest};
}
sub is_cached_common_sublink {
  my ($class,$sublink) = @_;
  return $CommonSubLinks{$sublink};
}
sub load_common_fingers_and_sublinks {
 my ($class, $dbh) = @_;

# my $dbh = DBI->connect("DBI:mysql:accessio_handprints:localhost"
#                        , "miavia_rdr"
#                        , $dbPasswd{"miavia_rdr"}
#                        , { RaiseError => 1 });
 my $queryStr = qq{ SELECT CommonFinger_digest_code
     FROM CommonFinger };

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 while (my ($digest_code) = $sth->fetchrow_array()) {
       $CommonFingers{$digest_code}++;
 }
 $sth->finish();

 $queryStr = qq{ SELECT CommonSubLink_sub_string
     FROM CommonSubLink };

 $sth = $dbh->prepare($queryStr);
 $sth->execute();
 while (my ($link_string) = $sth->fetchrow_array()) {
       $CommonSubLinks{$link_string}++;
 }
 $sth->finish();

# $dbh->disconnect();
 # return what??
}
sub load_common_sublinks {
 my ($dbh) = @_;

 my $queryStr = qq{ SELECT CommonSubLink_sub_string
     FROM CommonSubLink };

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 while (my ($link_string) = $sth->fetchrow_array()) {
       $CommonSubLinks{$link_string}++;
 }
 $sth->finish();

}
############ DB Open Subs ############
# open a db connection, using values from MiaVia
#
sub openDb {
  my $user = shift;
  my $dbh = DBI->connect("DBI:mysql:accessio_handprints:localhost"
                        , $user
                        , $dbPasswd{$user}
                        , { RaiseError => 1 });
  return $dbh;
}
# added 11-21-03 to support db modification across networked machines
sub openNetworkDb {
  my $user = shift;
  my $host = shift;
  my $dbh = DBI->connect("DBI:mysql:accessio_handprints:$host"
                        , $user
                        , $dbPasswd{$user}
```

```
                      , { RaiseError => 1 });
  return $dbh;
}
############ HashCache Subs ############
#
sub openHashCacheDb {
 my $user = shift;
 my $servername = shift;
 my $dbh = DBI->connect("DBI:mysql:accessio_hashcache:$servername" ,
  $user ,
  $dbPasswd{$user} ,
  { RaiseError => 1 }
 );
   return $dbh;
}
sub isCached {
  my $md5sum = shift;
  my $dbh = shift;
  my $score = 0;
  my $queryStr = qq{select Score FROM HashCache WHERE MD5SUM =};
  $queryStr .= "\"$md5sum\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my $numRows = $sth->rows;
  if( ($sth->rows) > 0) {
 ($score) = $sth->fetchrow_array();
  }
  return $score;
}
sub cacheId {
  my $md5sum = shift;
  my $dbh = shift;
  my $cacheId = 0;
  my $queryStr = qq{select MessageId FROM HashCache WHERE MD5SUM =};
  $queryStr .= "\"$md5sum\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my $numRows = $sth->rows;
  if( ($sth->rows) > 0) {
 ($cacheId) = $sth->fetchrow_array();
  }
  return $cacheId;
}
sub cacheCategory {
  my $md5sum = shift;
  my $dbh = shift;
  my $category = 0;
  my $queryStr = qq{select Category FROM HashCache WHERE MD5SUM =};
  $queryStr .= "\"$md5sum\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my $numRows = $sth->rows;
  if( ($sth->rows) > 0) {
 ($category) = $sth->fetchrow_array();
  }
  return $category;
}
sub stashHash {
 my $md5sum = shift;
 my $score = shift;
 my $msg_id = shift;
 my $category = shift;
 my $dbh = shift;
 my $insertStr = qq{ INSERT INTO HashCache (MD5SUM,Score,Messageid,Category)
  VALUES ("$md5sum","$score", "$msg_id","$category") };
 my $sth = $dbh->prepare($insertStr);
 $sth->execute();
 $sth->finish();
}
sub purgeCache {
 my $dbh = shift;
 # Delete everything older than 15 minutes with a score less than 40
 my $queryStr = qq{ DELETE from HashCache WHERE MsgTime < FROM_UNIXTIME(UNIX_TIMESTAMP() - (60 *
15) ) and Score < 40.00 };
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 my $numYoung = $sth->rows;
 # Delete everything older than 24 hours, regardless of score
 $queryStr = qq{ DELETE from HashCache WHERE MsgTime < FROM_UNIXTIME(UNIX_TIMESTAMP() - (60 * 1440)
) };
 $sth = $dbh->prepare($queryStr);
 $ret = $sth->execute();
 my $numOld = $sth->rows;
 $sth->finish();
 return ($numYoung,$numOld);
}
sub countHash {
 my $dbh = shift;
 my $count = 0;
 my $queryStr = qq{ select COUNT(MsgTime) as Count from HashCache };
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 my $numRows = $sth->rows;
 if( ($numRows) > 0)
 {
  ($count) = $sth->fetchrow_array();
 }
 return $count;
}
sub purgeCacheOptions {
 my $dbh = shift;
 my $young_mins = shift;
 my $old_mins = shift;
 my $score_threshold = shift;
 if (!(defined $young_mins)) {
  $young_mins=15;
 }
 if (!(defined $old_mins)) {
  $old_mins=1440;
 }

 if (!(defined $score_threshold)) {
  $score_threshold=40;
 }
 my $young_secs = $young_mins * 60;
 my $old_secs = $old_mins * 60;

 # Delete everything older than [young] minutes with a score less than [threshold]

 my $queryStr = qq{ DELETE from HashCache WHERE MsgTime < FROM_UNIXTIME(UNIX_TIMESTAMP() -
($young_secs) ) and Score < $score_threshold };
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 my $numYoung = $sth->rows;
 # Delete everything older than [old] minutes, regardless of score
 $queryStr = qq{ DELETE from HashCache WHERE MsgTime < FROM_UNIXTIME(UNIX_TIMESTAMP() -
($old_secs) ) };
 $sth = $dbh->prepare($queryStr);
 $ret = $sth->execute();
 my $numOld = $sth->rows;
 $sth->finish();
 return ($numYoung,$numOld);
}
############ Spam DB Getter Subs ############
#
sub getMessage_filename {
  my $msg_id = shift;
  my $dbh = shift;
  my $msg_filename;

  my $queryStr = qq{select Message_filename FROM Message WHERE Message_id =};
  $queryStr .= "\"$msg_id\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  ($msg_filename) = $sth->fetchrow_array();

  return $msg_filename;
}
sub getMessage_id {
  my $msg_filename = shift;
  my $dbh = shift;
  my $msg_id;

  my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
  $queryStr .= "\"$msg_filename\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  ($msg_id) = $sth->fetchrow_array();

  return $msg_id;
}
# added 11-21-03
# given a dbhandle and a reference to array of sub links, it returns a reference to a
# hash of wildlinks by string, with the value being a reference to an array of hashes
# of message_id/weight pairs
sub getWildLinks {
 my $dbh = shift;
 my $sublinkREF = shift;
       my @sublinks = @{$sublinkREF};
 my @WildLinkList = ();
 my @wildLinks=();
 my %WildLinksByString =();
 my @uncommon_sublinks=();


 if (scalar(@sublinks) > 0) {

  foreach my $sub_link (@sublinks) {
        #if (!(is_common_sublink($dbh,$sub_link))) {
        if (!($CommonSubLinks{$sub_link})) {
           push @uncommon_sublinks, $sub_link;
          }
  }

  # get a list of wildLink ids that match our sub string list in SubLinks
  my $queryStr = qq{select SubLink_WildLink_id FROM SubLink WHERE
    SubLink_sub_string IN };
  my $sub_string = shift @uncommon_sublinks;
  $queryStr .= "(\"$sub_string\"";
  foreach $sub_string (@uncommon_sublinks) {
       $queryStr .= ",\"$sub_string\"";
  }
  $queryStr .= ")";
  my $sth = $dbh->prepare($queryStr);
    my $ret = $sth->execute();
  my %seen = (); # so only unique WildLinks added to list
  while (my ($WildLinkId) = $sth->fetchrow_array()) {
       push (@WildLinkList, $WildLinkId) unless $seen{$WildLinkId}++;
  }
  # Now, from the list of WildLink ids, go get the rest of the WildLink data
  if (scalar(@WildLinkList) > 0) { # there are elements in the link list
   $queryStr = qq{select WildLink_link_string, WildLink_Message_id,
            WildLink_weight FROM WildLink WHERE WildLink_id IN };
   my $link_id = shift @WildLinkList;
     $queryStr .= "(\"$link_id\"";
   foreach $sub_string (@WildLinkList) {
         $queryStr .= ",\"$sub_string\"";
   }
   $queryStr .= ")";
   $sth = $dbh->prepare($queryStr);
     $ret = $sth->execute();
     while (my ($link_string, $msg_id, $weight) = $sth->fetchrow_array()) {
    if (!(defined $weight) || ($weight == 0)){
     $weight = 1;
    }
    $link_string =~ s/\s+//;
    $link_string =~ s/\r//;
    $link_string =~ s/\n//;
    push @wildLinks, ({string => $link_string, msgid => $msg_id,
     weight => $weight});
   }
  }

  # put the data (an array of hashes) into a usable format - a hash with the
  # string being the index, and the value being a reference to an array of
  # hashes that contain the message id and weight as the name/value pair.
  # (since the same wildcard string can be in more than one message, but we
  # can have different weights in different messages, but don't
  # want to test for a match against the same string multiple times)

  foreach my $wildLink (@wildLinks) {
   my $curr_string = 0;
   $curr_string = $wildLink->{string};
   my $curr_msgid = 0;
   $curr_msgid = $wildLink->{msgid};
   my $curr_weight = 1;
   $curr_weight = $wildLink->{weight};

   # create the new element
   my $new_element = {msgid => $curr_msgid, weight => $curr_weight};
         # add it to the array in the byString hash for this string
   push @{$WildLinksByString{$curr_string}},$new_element;
  }
 }

  return  \%WildLinksByString;
}
sub getFingerIds {
 my ($dbh, $msg_id) = @_;
```

```
   my %fingerIds;

 my $queryStr = qq{ SELECT Finger_id, Finger_digest_code
  FROM Finger WHERE Finger_Message_id='$msg_id'};

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 while (my ($finger_id, $digest) = $sth->fetchrow_array()) {
    $fingerIds{$finger_id} = $digest;
 }
 $sth->finish();

  return \%fingerIds;
}
# given a catgeory id, get the category string from the database
sub getCatString {
 my ($dbh, $cat_id) = @_;
   my $cat_string;

 my $queryStr = qq{ SELECT Category_name FROM Category
  WHERE Category_id='$cat_id'};

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 ($cat_string) = $sth->fetchrow_array();
 $sth->finish();

  return $cat_string;
}
# given a message id, get the category of the message from the database
sub getCategory {
 my ($dbh, $msg_id) = @_;
   my $cat_id;

 my $queryStr = qq{ SELECT Message_Category_id FROM Message
  WHERE Message_id='$msg_id'};

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 ($cat_id) = $sth->fetchrow_array();
 $sth->finish();

  return $cat_id;
}
############ Spam DB Modify Subs ############
#
# added 11-19-03
# this routine is intended to be used for modifying the values of fingers
# for example, to mark a finger as a noise finger (set the length to zero)
# , this routine should be called with the following parameters:
```

```
#     $where_field = 'Finger_id'
#     $where_value =  the finger id of the specific finger to modify
#     $change_field = 'Finger_length'
#     $change_value = '0'
sub updateFinger {
  my ($dbh, $where_field, $where_value, $change_field, $new_value) = @_;

  my $updateStr = qq { UPDATE Finger SET $change_field='$new_value'
                        WHERE $where_field='$where_value'
                     };
  my $sth = $dbh->prepare($updateStr);
  my $result = $sth->execute();
  $sth->finish();
  return $result;
}
sub deleteFinger {
  my ($dbh, $where_field, $where_value) = @_;

  my $deleteStr = qq{delete FROM Finger WHERE $where_field='$where_value'};
  my $sth = $dbh->prepare($deleteStr);
  my $ret = $sth->execute();
  $sth->finish();
  return $ret;
}
sub deleteMessage_id {
  my $msg_id = shift;
  my $dbh = shift;
  my $deleteStr;
  my $ret;
  my $sth;
  my @wildcard_ids;

  # remove message
  $deleteStr = qq{delete FROM Message WHERE Message_id=};
  $deleteStr .= "\"$msg_id\"";
  $sth = $dbh->prepare($deleteStr);
  $ret = $sth->execute();
  $sth->finish();
  # remove fingers
  $deleteStr = qq{delete FROM Finger WHERE Finger_Message_id =};
  $deleteStr .= "\"$msg_id\"";
  $sth = $dbh->prepare($deleteStr);
  $ret = $sth->execute();
  $sth->finish();
  # remove finger sets
  $deleteStr = qq{delete FROM Finger_set WHERE Finger_set_Message_id =};
  $deleteStr .= "\"$msg_id\"";
  $sth = $dbh->prepare($deleteStr);
  $ret = $sth->execute();
  $sth->finish();
```

```
# get wild links ids
my $queryStr = qq{select WildLink_id FROM WildLink WHERE WildLink_Message_id =};
$queryStr .= "\"$msg_id\"";
$sth = $dbh->prepare($queryStr);
$ret = $sth->execute();
while (my ($wild_id) = $sth->fetchrow_array()) {
  push @wildcard_ids, $wild_id;
}
# if there are wild links, remove them and their link sub string entries
if (scalar(@wildcard_ids) > 0) {
   my @sublink_ids;

 # get sublink ids for those wild links
 $queryStr = qq{ SELECT SubLink_id
 FROM SubLink WHERE SubLink_WildLink_id IN };
 my $sub_string = $wildcard_ids[0];
 $queryStr .= "(\"$sub_string\"";
 foreach $sub_string (@wildcard_ids) {
      $queryStr .= ",\"$sub_string\"";
 }
 $queryStr .= ")";
 $sth = $dbh->prepare($queryStr);
 $ret = $sth->execute();
 while (my ($sublink_id) = $sth->fetchrow_array()) {
   push @sublink_ids, $sublink_id;
 }

 # remove sub link entries
 $deleteStr = qq{delete FROM SubLink WHERE SubLink_id IN };
 $sub_string = shift @sublink_ids;
 $deleteStr .= "(\"$sub_string\"";
 foreach $sub_string (@sublink_ids) {
      $deleteStr .= ",\"$sub_string\"";
 }
 $deleteStr .= ")";
 $sth = $dbh->prepare($deleteStr);
 $ret = $sth->execute();


 # remove wild links
 $deleteStr = qq{delete FROM WildLink WHERE WildLink_id IN};
 $sub_string = shift @wildcard_ids;
 $deleteStr .= "(\"$sub_string\"";
 foreach $sub_string (@wildcard_ids) {
      $deleteStr .= ",\"$sub_string\"";
 }
 $deleteStr .= ")";
 $sth = $dbh->prepare($deleteStr);
 $ret = $sth->execute();
```

```
   $sth->finish();
  } # end if wild link ids exist
}
# given a message id, a wild card string, a reference to an array of sub links,
# and the weight for the wild link, insert the wild card link into the WildLink
# table, then insert it's sublinks (for lookup) into the SubLink table.  The
# array of sublinks should be the sub links included in the wildcard string.
# since this requires a human to determine the wildcard string, it is not
# part of the automated message insert process.
sub insertWildSubs {
 my $dbh = shift;
 my $msg_id = shift;
 my $wild_string = shift;
 my $sublinkREF = shift; # reference to sublinks array
 my $weight = shift;


  load_common_sublinks ($dbh);

  $wild_string =~ s/\s+//;
  $wild_string =~ s/\r//;
  $wild_string =~ s/\n//;
  # insert the wild link
  my $insertStr = qq { INSERT INTO WildLink
        (WildLink_link_string,  WildLink_message_id, WildLink_weight, WildLink_inserted_on)
                values ( '$wild_string', '$msg_id', '$weight', NOW())
                };
        my $sth = $dbh->prepare($insertStr);
 my $result = $sth->execute();

        # get the Wild Link ID
        my $getAutoIncStr = qq { SELECT LAST_INSERT_ID()};
   $sth = $dbh->prepare($getAutoIncStr);
   $sth->execute();
   my ($wildId) = $sth->fetchrow_array();
 # insert the sublinks for this wild link
 foreach my $sublink (@{$sublinkREF}) {
  if (!($CommonSubLinks{$sublink})) {
   my $insertStr = qq { INSERT INTO SubLink
          (SubLink_sub_string,  SubLink_WildLink_id, SubLink_inserted_on)
                 values ( '$sublink', '$wildId', NOW()) };
                my $sth = $dbh->prepare($insertStr);
   my $result = $sth->execute();
   #print "sublink insert result is $result\n";
               }

 }
 $sth->finish();
}
# added 11-11-02 by Liz Derr
# for deleting the entries in the message_queue database, but can be
# used for the main db, too depending on the dbh passed in
sub deleteMessage_filename {
  my $msg_filename = shift;
  my $dbh = shift;
  my $deleteStr;
  my $msg_id;
  my $ret;
  my $sth;

  # get the message id
  my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
  $queryStr .= "\"$msg_filename\"";
  $sth = $dbh->prepare($queryStr);
  $ret = $sth->execute();
  ($msg_id) = $sth->fetchrow_array();

  if ($msg_id) {
    deleteMessage_id ($msg_id, $dbh);
  }
}
############ Misc. Subs ############
# build a proper date-time-stamp string for MySQL entries
#
sub getDateTimeStamp {
  my ($sec,$min,$hr,$mday,$mon,$year,$wday,$yday,$isdst) = localtime;
  my $now = sprintf "%4d-%02d-%02d %02d:%02d:%02d",$year+1900,$mon+1,$mday,$hr,$min,$sec;
  return $now;
}
sub is_common_finger {
  my ($dbh, $finger_digest) = @_;

  my $finger_id=0;
  my $queryStr = qq{ SELECT CommonFinger_id
     FROM CommonFinger where CommonFinger_digest_code=};
  $queryStr .= "\"$finger_digest\"";

  my $sth = $dbh->prepare($queryStr);
  $sth->execute();
  ($finger_id) = $sth->fetchrow_array();
  $sth->finish();

  return $finger_id;  # will be 0 if not a common finger

}
sub is_common_sublink {
  my ($dbh, $link_string) = @_;

  my $sublink_id = 0;
  my $queryStr = qq{ SELECT CommonSubLink_id
     FROM CommonSubLink where CommonSubLink_sub_string=};
  $queryStr .= "\"$link_string\"";

  my $sth = $dbh->prepare($queryStr);
  $sth->execute();
  ($sublink_id) = $sth->fetchrow_array();
  $sth->finish();

  return $sublink_id;  # will be 0 if not a common finger

}
sub get_last_msgid {
 my $dbh = shift;

 my $queryStr = qq{ SELECT Message_id FROM Message order by Message_id DESC LIMIT 1};

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 my ($msg_id) = $sth->fetchrow_array();
 $sth->finish();

  return $msg_id;
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: HtmlTagProcessor.pm,v 1.76 2004/05/28 05:16:27 lizd Exp $
#
# 2-15-04 clear of all Settings references

package MiaVia::HtmlTagProcessor;
## Am HtmlTagProcessor collects relevant information from html tags
use strict;
use MiaVia::Utils;
## CLASS METHODS
##
sub new {
  my ($class) = @_;

  my @hrefList = ();
  my %hrefHash = ();
  my @mailtoList = ();
  my @imageList = ();
  my @linksubList = ();

  my $self = { hrefList => \@hrefList
        , hrefHash => \%hrefHash
        , mailtoList => \@mailtoList
        , imageList => \@imageList
        , linksubList => \@linksubList};

  bless $self, $class;
}
## INSTANCE METHODS
##
sub add {
  my ($self, $type, $value) = @_;
  if ($type eq 'href') {
    my ($preface,$after) = split /:/, $value;
    if ($preface eq 'mailto') {
       push @{$self->{mailtoList}}, $after;
    } else { # assume it's a URL
       #print "tag href: type: $type, preface: $preface, after: $after, value: $value\n";
     my $newurl = $self->cleanUrl($value);
     if ($newurl ne "href") { #href as an URL is useless
      push @{$self->{hrefList}}, $newurl;
      $self->getLinkSubs($newurl);
     }
   } # end else it's a URL
  } elsif ($type eq 'src') {
   #my ($preface,$after) = split /:/, $value;
   #print "tag src: type: $type, preface: $preface, after: $after, value: $value\n";
     my $newurl = $self->cleanUrl($value);
     if ($newurl ne "src") { #src as an url is useless
    push @{$self->{imageList}}, $newurl;
    $self->getLinkSubs($newurl);
   }
  } else {
   if ($type =~ /http/) {
    my $newurl = $self->cleanUrl($type);
  if ($newurl ne "href") {
      push @{$self->{hrefList}}, $newurl;
      $self->getLinkSubs($newurl);
   }
 } elsif ($value =~ /http/) {
  my $newurl = $self->cleanUrl($value);
  if ($newurl ne "href") {
      push @{$self->{hrefList}}, $newurl;
      $self->getLinkSubs($newurl);
   }
 }
   #print "unknown attribute $type has value $value\n";
  }
}
sub href_list {
  my $self = shift;
  return $self->{hrefList};
}
sub href_hash {
  my $self = shift;
  return $self->{hrefHash};
}
sub mailto_list {
  my $self = shift;
  return $self->{mailtoList};
}
sub image_list {
  my $self = shift;
  return $self->{imageList};
}
sub linksub_list {
  my $self = shift;
  return $self->{linksubList};
}
sub cleanUrl {
    my ($self, $url) = @_;

    #print "cleaning URL - before $url\n";

    my $newurl = $url;

    my $normalized_url = MiaVia::Utils::normalize_url($newurl);
    $newurl=$normalized_url;
 $newurl =~ s/\"//g;
 $newurl =~ s/\'//g;
 $newurl =~ s/\`//g;


  # strip trailing cgi params
  if ($newurl =~ /\?/) {
   my @pieces = split /\?/, $newurl;
   my $num = 0;
   $num = scalar(@pieces);
   #print "num scalar is $num\n";
   #if ($num > 2) {
   if ($num > 1) {
    my $iter = 1;
    $newurl = $pieces[0]; # assume normal link with CGI params
    while ($iter < $num) {
     if  ($pieces[$iter] =~/http/) { # it looks like a redirect
      $newurl = $pieces[$iter]; #so get the redirected URL instead
     }
     $iter++;
    }
        #$newurl = $pieces[$num-2]; # get the second to last piece
       #} else {
        #$newurl = shift(@pieces);
       #}
   }
  }
  # strip noise after #
  if ($newurl =~ /\#/) {
   my @pieces = split /\#/, $newurl;
       my $base = shift @pieces;
       $newurl = $base;
  }
  # strip redirect before *
  if ($newurl =~ /\*/) {
   my @pieces = split /\*/, $newurl;
       my $count = scalar @pieces;
   my $base = $pieces[$count-1];
       $newurl = $base;
  }
  #strip pre-@ noise in domain only
  if ($newurl =~ /\@/) {
    if ($newurl =~ /http:\/\//) {
     $newurl =~ s/http:\/\///;
    }
# first, get the domain
my @levels = split "/", $newurl;
my $domain = shift @levels;
```

```
  # next, look for spurious preceeding @s in the base
  my @noises = split /\@/, $domain;
  my $count = scalar @noises;
  my $newdomain = $noises[$count-1];

  # construct the new url from the cleansed base and remaining levels
  $newurl = "http://".$newdomain; # start fresh
  foreach my $level (@levels) {
   $newurl = $newurl."/".$level;
  }
    }

    # lowercase the URL - added 6/24/03
    $newurl =~ tr/A-Z/a-z/;
    #print "final URL is $newurl\n";
    #print "cleaning URL - after $newurl\n";
    return $newurl;

}
sub getLinkSubs {
   my ($self, $url) = @_;

   $url =~ s/http:\/\///;
   $url =~ s/\"//g;
   $url =~ s/\'//g;
   $url =~ s/\`//g;
   my @subList = ();
   my @moresubs = ();

   my @linkSubs = split "/", $url;

   foreach my $mainsub (@linkSubs) {
    @moresubs = ();
    @moresubs = split /\./, $mainsub;
    push @subList, @moresubs;
   }

   foreach my $linksub (@subList)  {
      push @{$self->{linksubList}}, $linksub;
   }

}
1; # end of HtmlTagProcessor.pm
```

```
######################################################################
# create the spam database and users
CREATE DATABASE accessio_handprints;
GRANT ALL ON accessio_handprints.* TO dba@localhost IDENTIFIED BY "XXXXX";
GRANT SELECT ON accessio_handprints.* TO miavia_rdr@localhost
  IDENTIFIED BY "XXXXX";
GRANT SELECT, INSERT, UPDATE, DELETE ON accessio_handprints.* TO
  miavia_wrtr@localhost IDENTIFIED BY "XXXXX";

######################################################################
# create the queued message and non-spam database
CREATE DATABASE queued_messages;
GRANT ALL ON queued_messages.* TO dba@localhost IDENTIFIED BY "XXXXX";
GRANT SELECT ON queued_messages.* TO miavia_rdr@localhost
  IDENTIFIED BY "XXXXX";
GRANT SELECT, INSERT, UPDATE, DELETE ON queued_messages.* TO
  miavia_wrtr@localhost IDENTIFIED BY "XXXXX";
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#  $Id: PipelineLogEvent.pm,v 1.8 2004/05/25 07:27:23 lizd Exp $
#
#   Revision History
#   ---------------
#   07-07-02  Ramon Created
#   07-09-02  Ramon initial check in to CVS
#   07-21-02  Liz Derr - added filename
#   07-24-02  Liz Derr - added subject, sender, mail_date
#                       and mail_message_id for CC processing
#                       added register type and logCCEvent sub
#   07-25-02  Liz Derr - added Subject to event log format
#                       removed hour designation from log file names
#   09-03-02  Liz Derr - added call to call_for_help instead of die
#   10-24-02  Liz Derr - added crono-opts for daemon logs
#    3-17-04  updated for pipeline use only
# used by pipeline ONLY
package PipelineLogEvent;
use strict;
use Exporter;
use MiaVia::Utils;
our @ISA = qw(Exporter);
our @EXPORTS = qw(
                new
                logEvent
                FilterVersion
                LogFormat
                DBVersion
                makeTimeStamp
                );
my %logidentifier_to_hashkey =  (
 t => 'TIMESTAMP'
 , I => 'FILTER_INSTANCE'
 , V => 'FILTER_VERSION'
 , c => 'DB_VERSION'
 , h => 'EMAIL_ID'
 , s => 'SPAMICITY'
 , x => 'DB_MSG_ID_LIST'
 , S => 'STATUS'
 , f => 'FILENAME' # added 7-21-02 by Lizd
 , F => 'FROM'
 , T => 'TO'
 , b => 'SUBJECT' # added 7-24-02 by Lizd
 , n => 'SENDER' # added 7-24-02 by Lizd
 , d => 'MAIL_DATE' # added 7-24-02 by Lizd
 , m => 'MAIL_MSG_ID' # added 7-24-02 by Lizd
 , Z => 'SIZE'
 , C => 'COMMENT'
```

```
);
#
#   LogEvent contstructor
#
#   initializes a few "Log" oriented variables.
sub new {
  my $class = shift;
  my %myargs = @_;      # slurp up the rest of the args...
  my $self = {};
  $self->{SIZE} = 0;
  $self->{TYPE} = ( exists $myargs{TYPE} ) ? $myargs{TYPE} : "filter";
  my $default_logformat = "%t %l %V %c %h %s %x %S %f %F %T %b %n %d %m %Z %C";
  # somehow code a path to cronolog
  my $path_to_cronolog = "mvlogger";
  #  set some globals...
  $self->{FILTER_VERSION} =
      (exists $myargs{FILTER_VERSION}) ? $myargs{FILTER_VERSION} : 4;
  $self->{FILTER_INSTANCE} =
      (exists $myargs{FILTER_INSTANCE}) ? $myargs{FILTER_INSTANCE} : 1;
  $self->{LOGFORMAT} =
      (exists $myargs{LOGFORMAT}) ? $myargs{LOGFORMAT} : $default_logformat;
  $self->{PATH_TO_CRONOLOG} =
      (exists $myargs{PATH_TO_CRONOLOG}) ? $myargs{PATH_TO_CRONOLOG} : $path_to_cronolog;
  $self->{LOG_DIR} = (exists $myargs{LOG_DIR} ) ? $myargs{LOG_DIR} : "/tmp/";

  #  - see how we're called an initialize the correct log file...
  my $cronoopts = "%Y-%d-%m-accessio-filter.log";
  my $mytype = $self->{TYPE};
  if ( $mytype eq "register" ) {
    $cronoopts = "%Y-%d-%m-register-spam.log";
  } elsif ( $mytype eq "fpos" ) {
   $cronoopts = "%Y-%d-%m-fpos_report.log";
  } else {
    $cronoopts = "none";
  }

  $cronoopts = $self->{LOG_DIR} . $cronoopts;

  # --- open up cronolog and set the global file handle..
  my $rc = open(my $localFH, "| $self->{PATH_TO_CRONOLOG} $cronoopts" );
  $self->{LOGFH} = $localFH;
  bless $self, $class;
  return $self;
}
# - Normal way of creating a time stamp.
sub makeTimeStamp {
  my $self = shift;
  my @thistime = localtime();
  my $realyear = 1900 + $thistime[5];
  my $realmonth = $thistime[4] + 1;
  my $timestamp = sprintf ( "%4d-%02d-%02d %02d:%02d:%02d",
      $realyear, $realmonth, $thistime[3], $thistime[2], $thistime[1], $thistime[0] );
  return $timestamp;
}
####################################################
# this is to log events from the processing of incoming
# spam to the command center
sub NewlogCCEvent {
  my $self = shift;
  my %myargs = @_;
  my $localFH = $self->{LOGFH};
  my $logStr = "";
  # - set the output sequence...
  my @outputSequence = (
 "TIME_STAMP",
 "STATUS",
 "COMMENT",
 "MAIL_DATE",
      "SENDER",
      "FROM",
      "TO",
      "SUBJECT",
      "MAIL_MSG_ID"
#       "DB_MSG_ID_LIST",
#       "FILENAME",
#       "FILTER_VERSION"
  );
  my $tempcomment = $myargs{ "COMMENT" };
  if ( index( $tempcomment, "\"" ) != 0 ) {
    $tempcomment = "\"" . $tempcomment . "\"";
  }
  $myargs{ "COMMENT" } = $tempcomment;
  my $curoutputvar;
  if ( !exists $myargs{TIME_STAMP} ) {
    $myargs{TIME_STAMP} = makeTimeStamp();
  }
  if ( !exists $myargs{FILTER_VERSION} ) {
    $myargs{FILTER_VERSION} = $self->{FILTER_VERSION};
  }
  foreach $curoutputvar ( @outputSequence ) {
    if ( exists $myargs{ $curoutputvar } ) {
      $logStr .= $myargs{ $curoutputvar } . " ";
    } else {
      $logStr .= "- ";
    }
  }
  print $localFH $logStr . "\n";
}
####################################################
# this is to log events from the processing of incoming
# spam to the command center
sub NewlogFposEvent {
  my $self = shift;
  my %myargs = @_;
  my $localFH = $self->{LOGFH};
  my $logStr = "";
  # - set the output sequence...
  my @outputSequence = (
 "TIME_STAMP",
 "STATUS",
      "TO",
      "MAIL_DATE",
      "SENDER",
      "FROM",
      "SUBJECT",
      "MAIL_MSG_ID",
      "FILENAME"
  );
  if ( !exists $myargs{TIME_STAMP} ) {
    $myargs{TIME_STAMP} = makeTimeStamp();
  }
  my $curoutputvar;
  foreach $curoutputvar ( @outputSequence ) {
    if ( exists $myargs{ $curoutputvar } ) {
      $logStr .= $myargs{ $curoutputvar } . " ";
    } else {
      $logStr .= "- ";
    }
  }
  print $localFH $logStr . "\n";
}
##################################################
sub Type {
  my $self = shift;
  if ( @_ ) {  $self->{TYPE} = shift; }
  return $self->{TYPE};
}
sub LogFormat {
  my $self = shift;
  if ( @_ ) {  $self->{LOGFORMAT} = shift; }
  return $self->{LOGFORMAT};
}
sub FilterVersion {
  my $self = shift;
  if ( @_ ) {  $self->{FILTERVERSION} = shift; }
  return $self->{FILTERVERSION};
}
##################################################
# this is to log events from the processing of incoming
# spam to the command center
```

```
sub logCCEvent {
  my $self = shift;
  my %myargs = @_;
  my $localFH = $self->{LOGFH};
  my $logStr = "";
  # - set the output sequence...
  my @outputSequence = (
 "TIME_STAMP",
      "SENDER",
      "FROM",
      "TO",
      "SUBJECT",
      "MAIL_DATE",
      "MAIL_MSG_ID",
      "STATUS",
      "DB_MSG_ID_LIST",
      "FILENAME",
      "COMMENT",
      "FILTER_VERSION",
      "DB_VERSION"
  );
  my $tempcomment = $myargs{ "COMMENT" };
  if ( index( $tempcomment, "\"" ) != 0 ) {
    $tempcomment = "\"" . $tempcomment . "\"";
  }
  $myargs{ "COMMENT" } = $tempcomment;
  my $curoutputvar;
  if ( !exists $myargs{TIME_STAMP} ) {
    $myargs{TIME_STAMP} = makeTimeStamp();
  }
  if ( !exists $myargs{FILTER_VERSION} ) {
    $myargs{FILTER_VERSION} = $self->{FILTER_VERSION};
  }
  if ( !exists $myargs{DB_VERSION} ) {
    $myargs{DB_VERSION} = $self->{DB_VERSION};
  }
  foreach $curoutputvar ( @outputSequence ) {
    if ( exists $myargs{ $curoutputvar } ) {
      $logStr .= $myargs{ $curoutputvar } . " ";
    } else {
      $logStr .= "- ";
    }
  }
  print $localFH $logStr . "\n";
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#  $Id: PipelineQueueMimeSpam.pm,v 1.3 2004/03/26 09:03:14 lizd Exp $
#
#
#
# this code was extracted from registerFOlderSpam.pl
#
# it extracts embedded mail messages and processes them
# as individual messages, if they exist.  This allows
# for people to forward spam as an attachment so that
# we can process the original spam and not the message
# that the user sends to us (i.e. the "reporting" envelope
# around the spam).  It assumes that if there are embedded
# messages, that the containing message is not to be
# processed.
#
# created 11-09-02 by Liz Derr replaces ProcessMimeSpam for
#    new Command Center
#  3-17-04 updated for pipeline changes
#############PIPELINE###############
package MiaVia::PipelineQueueMimeSpam;
use MIME::Parser;
use MIME::Entity;
use MiaVia::Message;
use MiaVia::PipelineEventRegisterSpam;
use MiaVia::PipelineSettings;
#############################################################
##
## calls eventRegisterSpam
##
#############################################################
sub process_message {
  my ($msg, $type) = @_;
  my   $embedded_msg;
   if ($type eq 'rfc822') {
    if (($msg->stringify_header =~ /Received:/) &&
        ($msg->stringify_body ne "")) {
         # we have a real message with a real header here
         PipelineEventRegisterSpam::queueSpam($msg);
        } elsif ($msg->stringify_body ne "") { # try and parse the body
             # print "gonna try and parse body \n";
             my $parser = MIME::Parser->new();
             $parser->output_dir($tmpDir);
             $parser->output_to_core(0);
             $parser->tmp_to_core(0);
             $parser->use_inner_files(0);
             eval { $embedded_msg = $parser->parse_data($msg->stringify_body);};
             if ($@) {
```

```
             # print "well, the body wouldn't parse\n";
             PipelineEventRegisterSpam::queueSpam($msg);
            } else {
             # print "body parsed!\n";
               PipelineEventRegisterSpam::queueSpam($embedded_msg);
            }
            $parser->filer->purge;
       } # end else try to parse

   } elsif ($msg->stringify_body =~ /Received:/) {
         # print "gonna try and parse body \n";
            my $parser = MIME::Parser->new();
            $parser->output_dir($tmpDir);
            $parser->output_to_core(0);
            $parser->tmp_to_core(0);
            $parser->use_inner_files(0);
            eval { $embedded_msg = $parser->parse_data($msg->stringify_body);};
            if ($@) {
             # print "well, the body wouldn't parse\n";
             PipelineEventRegisterSpam::queueSpam($msg);
            } else {
             # print "body parsed!\n";
               PipelineEventRegisterSpam::queueSpam($embedded_msg);
            }
            $parser->filer->purge;
    } else {
    PipelineEventRegisterSpam::queueSpam($msg);
   }


}
############################################################
##
##Recursion to step through nested multipart messages
##
############################################################
sub process_multipart_message {
  my ($mess, $level, $piece_number) = @_;
  my $msg_processed = 'N';
  foreach my $part ($mess->parts) {

     my $efftype = $part->effective_type;
     #print "$level effective type of the part is $efftype\n";
     # recursion to handle nesting
     if ($part->is_multipart) {
       $level++;
      #print "found a nested multi-part message\n";
       # call self recursively
       process_multipart_message ($part, $level, $piece_number);
     } #end if multi-part
    #not a multi-part part, because recursion wouldn't get here if it was
    if ($efftype =~ /822/) {
       my $numparts = $part->parts;
         if ($numparts > 1) {
          #print "found an rfc822 with many parts\n";
          $level++;
          process_multipart_message ($part, $level, $piece_number);
         } else {
            #print "calling process rfc822 message from multipart\n";
            process_message($part,"rfc822");
            $piece_number++;
            $msg_processed = 'Y'; #we processed extracted messages
         }
    } #end if $efftype
  } # end foreach part


  if ($msg_processed eq 'N') {# we didn't extract anything, so process whole msg
   #print "processed multi-part message as text SKELETON:\n";
   #$mess->dump_skeleton(\*STDOUT);
   process_message($mess, "text");
   $piece_number++;
   $level--;
  }
  return($piece_number);
}
1;
```

```
#!/bin/bash
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#
DATECMD=/bin/date
DAYDATE=`"$DATECMD" "+%Y-%d-%m"`
TIMESTAMP=`"$DATECMD" "+%Y-%m-%d"`
LOGFILE=/usr/local/accessio/log/${DAYDATE}-fpos_report.log
TOTALSUP=`grep -c "support@miavia.com" $LOGFILE`
TOTALBLARG=`grep -c "abuse@blarg.net" $LOGFILE`
TOTALUSER=`grep -c "spamassfp@usermail.com" $LOGFILE`
TOTAL=`grep -c $TIMESTAMP $LOGFILE`
echo "${TOTALSUP} sent to support. ${TOTALBLARG} sent to blarg. ${TOTALUSER} sent to usermail. ${TOTAL}
total false positive messages processed" | mail -s "${DAYDATE} Daily False Positive Summary"
insert_report@miavia.com
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: NewMessageFactory.pm,v 1.124 2004/05/14 23:11:33 lizd Exp $
#
# modified 9-24-02 by Liz Derr to support multiple fingers, this now obsoletes
#                     MessageFactory.pm
package MiaVia::NewMessageFactory;
## MessageFactory processes a MessageSource, creates a Message
## as appropriate for the type of MessageSource, returns the
## Message.
##
## Call this instead of the Message constructor!
#
# modified 8-21-02 by Liz Derr to add paragraph fingers
# modified 9-24-02 by Liz Derr to support multiple fingers, this now obsoletes MessageFactory.pm
# modified 10-29-02 by Liz Derr added non_text finger
# modified 1-17-03 by Liz Derr commented out non-text finger
# modified 11-23-03 by Liz Derr fixed bug with null HTML fingers and
#   made HTML fingers separate for each link, rather than concatenated
#   also added check for non-null body finger before creating finger
# 6-9-03 rolled back changes in version 1.69 that removed recip name removal
#       commented out body finger creation
# 6-15-03 added call to normalize_url for href fingers.
# 9-??-03 modified to determine dishonest mime types
# 11-19-03  added default weight to finger creation
#          added link sub fingers
# 2-15-04 clear of Settings references
use MIME::Parser;
use MiaVia::NewFinger;
use MiaVia::NewFingerSet;
use MiaVia::SubLink;
use MiaVia::Message;
use MiaVia::Utils;
use MiaVia::FingerSpec;
use MiaVia::NewMimeProcessor;
use MIME::QuotedPrint;
use Exporter;
# our @ISA = qw(Exporter);
use MiaVia::NewTextBodyProcessor;
use MiaVia::NewHtmlBodyProcessor;
## DATA
##
our  $version = "6.0";
# our @EXPORT = qw($version);
my $fingerSpec = MiaVia::FingerSpec->new({
    'text' => \&determineRealTextType
  , 'text/plain' => \&determineRealTextType
  , 'text/html' => \&determineRealTextType
#     'text' => \&procTextPlain # commented out 9-21-03
#   , 'text/plain' => \&procTextPlain # commented out 9-21-03
#   , 'text/html' => \&procTextHtml # commented out 9-21-03
#   , 'application' => \&procNonText # added 10-29-02
#   , 'image' => \&procNonText # added 10-29-02
#   , 'audio' => \&procNonText # added 10-29-02
#   , 'video' => \&procNonText # added 10-29-02
   });
##
## SUBROUTINES
sub createMessage {
  my $msgSource = shift;
  my $srcType = $msgSource->{type};

  if ($srcType eq 'mimeFile') {
    my $mp = MiaVia::NewMimeProcessor->new($msgSource);
    my $fingerSet = $mp->extractFingers($fingerSpec);
    my $message = MiaVia::Message->new($version, $fingerSet,
                                   { mimeProcessor => $mp
                                   , fileName => $msgSource->fileName()
                                   , recipient => $mp->recipName()
                                   , path => $mp->fileDate()
                                   });
    #$mp->cleanup(); bad idea - caused accessio to fail
    return $message;
  } elsif ($srcType eq 'mimeString') {
    my $mp = MiaVia::NewMimeProcessor->new($msgSource);
    my $fingerSet = $mp->extractFingers($fingerSpec);

    my $message = MiaVia::Message->new($version, $fingerSet,
                                   { mimeProcessor => $mp
                                   , recipient => $mp->recipName()
                                   });
    #$mp->cleanup();
    return $message;
  } elsif ($srcType eq 'smtp') {
    my $mp = MiaVia::NewMimeProcessor->new($msgSource);
    my $fingerSet = $mp->extractFingers($fingerSpec);

    my $smtpEnvelope = $msgSource->{smtpEnvelope};
    my $message = MiaVia::Message->new($version, $fingerSet,
                                   { mimeProcessor => $mp
                                   , smtpEnvelope => $smtpEnvelope
                                   , fileName => $msgSource->fileName()
                                   , recipient => $mp->recipName()
                                   , path => $mp->fileDate()
                                   });
    return $message;
  } else {
    return 1; #12-2-03 die isn't good here!
    #print STDERR "Cannot create Message of type '$srcType'\n";
    #die;
  }
}
sub procTextHtml {
  my $entity = shift;
  # note: $entity is a Mime Entity
  my $done=0;
  my $bodyProc = MiaVia::NewHtmlBodyProcessor->new();
  my $bodyFinger=0;
  my $recipName;
  my $recipDomain;
  my $recipFull = $entity->get('To');
  if (defined($recipFull)) {
   my ($recipDescr, $recipAddr) = split /</, $recipFull, 2;
   ($recipName, $recipDomain) = split /@/, $recipFull, 2;

   }

  my $mybh = $entity->bodyhandle();
  my $body_text = $mybh->as_string;
  my $decoded = MIME::QuotedPrint::decode_qp($body_text);
  my $noiseless_body = MiaVia::Utils::strip_html_noise($decoded);
  $bodyProc->add($noiseless_body);
  $bodyProc->done();   # flushes text buffer
  my $nested_html = $bodyProc->getNested();
  while ($nested_html) { # there is nested html
   $bodyProc->resetNested(); # reset the nested to blank
   $decoded = MIME::QuotedPrint::decode_qp($nested_html);
   $noiseless_body = MiaVia::Utils::strip_html_noise($decoded);
   $bodyProc->add($noiseless_body); # parse the nested html
   $nested_html = $bodyProc->getNested(); # see if there is any more nesting
   $bodyProc->done();  # flushes text buffer
  }

  my $html_sublink_REF = $bodyProc->linksubs();
  my $text_sublink_REF =  $bodyProc->{textProcessor}->linksubs();

  push (@{$html_sublink_REF}, @{$text_sublink_REF}); #add text link subs to html link sub list
  my $SubLinkElem = MiaVia::SubLink->new( $html_sublink_REF);

  # Create Paragraph Fingers
  my $count = 0;
  my @paragraphs=();
  my $paragraph = "";
  my @paragraphFingers =();
  my $paragraphFinger;
  my $paragraphSet=0;
  @paragraphs = @{$bodyProc->paragraphs()};
  foreach $paragraph (@paragraphs) {
    $count++;
   if (defined($recipName)) {
    my $safe_recipName= quotemeta($recipName);
    $paragraph =~ s/$safe_recipName//g;

   }

   # create a finger for each paragraph
   $paragraphFinger
      = MiaVia::NewFinger->new('stripped_paragraph'
                          , 2 #version, changed 11-21-02, less stripping - needed to add new version to NewFinger.pm, too!
                          , { 'value' => $paragraph }
                          , { 'weight' => 1 }  # the default weight 11-19-03
                          );
   push @paragraphFingers , $paragraphFinger;
} # end foreach paragraph

# create a paragraph finger set if there are paragraph fingers
if ($count > 0) {
 $paragraphSet = MiaVia::NewFingerSet->new(@paragraphFingers);
}

#######################################################
# create link fingers in these sections below
my ($link_finger, $mailto_finger, $image_finger);

# create href fingers
my $safe_pattern;
my $href_string ="";
my @hrefFingers =();
$count = 0;
my $hrefSet=0;
my @hrefs = @{$bodyProc->hrefs()}; # get the list of href strings

my @text_hrefs = @{$bodyProc->{textProcessor}->hrefs()};

push (@hrefs, @text_hrefs);
foreach my $href (@hrefs) {
 $safe_pattern = quotemeta($href); # for compare, in case there are weird chars in the string
 if (length($href) > 0) { # if not blank
  if (!($href_string =~ /$safe_pattern/)) { #if not already in the list
   $count++;
    $href_string = $href_string . $href;

    $link_finger = MiaVia::NewFinger->new('href_tag'
                          , 1 #version
                          , { 'value' => $href }
                          , { 'weight' => 1 }  # the default weight 11-19-03
                          );
              push @hrefFingers , $link_finger;
   }
```

```
}
 }
 # create an href finger set if there are href fingers
 if ($count > 0) {
  # print "creating set with $count href fingers\n";
  $hrefSet = MiaVia::NewFingerSet->new(@hrefFingers);
 }


 # create mailto fingers
 $count = 0;
 my $mailto_string ="";
 my @mailtoFingers =();
 my $mailtoSet=0;
 my @mailtos = @{$bodyProc->mailtos()}; # get the list of mailto strings
 foreach my $mailto (@mailtos) {
   $safe_pattern = quotemeta($mailto); # for compare, in case there are weird chars in the string
   if (length($mailto) > 0) { # if not blank
    if (!($mailto_string =~ /$safe_pattern/)) { # dedupe the mailto list
     $count++;
     $mailto_string = $mailto_string . $mailto;

     $mailto_finger = MiaVia::NewFinger->new('mailto_tag'
                         , 1 #version
                         , { 'value' => $mailto }
                         , { 'weight' => 1 }  # the default weight 11-19-03
                         );
                    push @mailtoFingers , $mailto_finger;
              }
   }
 }
 # create a mailto finger set if there are mailto fingers
 if ($count > 0) {
  $mailtoSet = MiaVia::NewFingerSet->new(@mailtoFingers);
 }

 # create image fingers
 $count = 0;
 my $image_string ="";
 my @imageFingers=();
 my $imageSet = 0;
 my @images = @{$bodyProc->images()}; # get list of image strings
 foreach my $image (@images) {
  $safe_pattern = quotemeta($image); # for compare, in case there are weird chars in the string
  if (length($image) > 0) { # if not blank
   if (!($image_string =~ /$safe_pattern/)) {
    $count++;
     $image_string = $image_string . $image;

     $image_finger = MiaVia::NewFinger->new('image_tag'
                    , 1 #version
                    , { 'value' => $image }
                    , { 'weight' => 1 }  # the default weight 11-19-03
                    );
                push @imageFingers , $image_finger;
      }
   }
   }
  # create an image finger set if there are image fingers
  if ($count > 0) {
   #print "creating set with $count image fingers\n";
   $imageSet = MiaVia::NewFingerSet->new(@imageFingers);
  }

  #print "creating whole finger set, body, paraset, hrefset, mailtoset, imageset\n";
  # return a fingerset that contains the stripped body,
  # the paragraph finger set, and the link sets
  my $whole_finger_set = MiaVia::NewFingerSet->new( #$bodyFinger, #commented out 6-9-03
    $paragraphSet, $hrefSet, $mailtoSet, $imageSet, $SubLinkElem);

  return $whole_finger_set;
}
sub procTextPlain {
  my $entity = shift;
  # note: $entity is a MIME entity

  my @paragraphs;
  my $paragraph = "";
  my @paragraphFingers;
  my $paragraphFinger;
  my $done=0;
  my $recipName;
  my $recipDomain;
  my $paraSet;
  my $linksubSet; #11-19-03
  my $whole_finger_set;

  my $bodyProc = MiaVia::NewTextBodyProcessor->new(1);

  my $recipFull = $entity->get('To');
  if (defined($recipFull)) {
   my ($recipDescr, $recipAddr) = split /</, $recipFull, 2; #added 6-21-03
   if (defined($recipAddr)) {
    if ($recipFull =~ /\@/) {
     ($recipName, $recipDomain) = split /@/, $recipAddr, 2;
    } else {
     $recipName=$recipAddr;
     $recipDomain='';
    }
   }
  if (defined($recipName)) {
   $recipName =~ s/\<//g; # commented out 6-21-03, added back in 7/3/03

  }
  }
 my $body = $entity->bodyhandle();
 $IO = $body->open("r");
 while (defined(my $line = $IO->getline)) {
    my $decoded = MIME::QuotedPrint::decode_qp($line);
    $bodyProc->add($decoded);

    # collect up stripped paragraphs
    ($paragraph, $done) = MiaVia::NewTextBodyProcessor::collectParagraph($line,$paragraph);
    if ($done == 1) {
   # if we're done with this para reset done,
   # add to list of paras, and clear paragraph.

   if (defined($recipName)) {
    my $safe_recipName= quotemeta($recipName);
    $paragraph =~ s/$safe_recipName//g;
   }
   push @paragraphs, $paragraph;
   $done = 0; #reset
   $paragraph =""; # clear out paragraph
}
 }
 $IO->close();
 foreach $paragraph (@paragraphs) {
    # create a finger for each paragraph
    $paragraphFinger
       = MiaVia::NewFinger->new('stripped_paragraph'
                          , 2 #version, changed 11-21-02, less stripping - needed to add new version to NewFinger.pm, too!
                          , { 'value' => $paragraph }
                          , { 'weight' => 1 }  # the default weight 11-19-03
                          );
    push @paragraphFingers , $paragraphFinger;
    } # end foreach paragraph

  # 8/28/03 create link fingers
  my $safe_pattern;
  my $href_string ="";
  my @hrefFingers =();
  my $count = 0;
  my $hrefSet=0;
  my @hrefs = @{$bodyProc->hrefs()}; # get the list of href strings

  foreach my $href (@hrefs) {
   #print "text href finger is $href\n";
  $safe_pattern = quotemeta($href); # for compare, in case there are weird chars in the string
  if (length($href) > 0) { # if not blank
```

```
    if (!($href_string =~ /$safe_pattern/)) { #if not already in the list
     $count++;
      $href_string = $href_string . $href;
      $link_finger = MiaVia::NewFinger->new('href_tag'
                              , 1 #version
                              , { 'value' => $href }
                              , { 'weight' => 1 }  # the default weight 11-19-03
                              );
                push @hrefFingers , $link_finger;
     }
  }
   } # end for each



   if ($count > 0) {# create an href finger set if there are href fingers
    my $SubLinkElem = MiaVia::SubLink->new($bodyProc->linksubs()); # 12-3-03
    $hrefSet = MiaVia::NewFingerSet->new(@hrefFingers);
    $paraSet = MiaVia::NewFingerSet->new(@paragraphFingers);
    #$linksubSet = MiaVia::NewFingerSet->new(@linksubFingers); # 11-19-03
    $whole_finger_set = MiaVia::NewFingerSet->new($paraSet, $hrefSet, $SubLinkElem); #, $linksubSet);
    } else { # no link or link sub fingers
    $whole_finger_set = MiaVia::NewFingerSet->new(@paragraphFingers);
    }
    return ($whole_finger_set);



}
# added 10-29-02 by Liz Derr
# this creates a finger for non-text Mime entites such as image
# or application attachments, among others
sub procNonText {
  my $entity = shift;
  my $finger = MiaVia::NewFinger->new('non_text'
                             , 1 #version
                             , { 'value' => $entity->stringify_body() }
                             , { 'weight' => 1 }  # the default weight 11-19-03
                             );
   # create a finger set
   return MiaVia::NewFingerSet->new($finger);
}
sub determineRealTextType { # created 9-21-03
 # this routine was created to detect messages that are "dishonest" about
 # thier mime type - specifically, if they say they are html but are actally
 # plain text, or they say they are plain text and are actually HTML
 # this looks for the presence of a <html> tag in the body of the message/part
 # if present, the message is determined to be HTML, otherwise the message is
 # determined to be plain text

 my $entity = shift;
 my $done = 0;
 my $is_html = 0;
 my $count = 0;

 my $body = $entity->bodyhandle();
   $IO = $body->open("r"); # commented out 6-15-03
   while (defined(my $line = $IO->getline) && ($done == 0)) {
      $count++;
      $line =~ tr/A-Z/a-z/;
     if (($line =~ /<html>/) ||
         ($line =~ /text\/html/) ||
         ($line =~ /<p/) ||
         ($line =~ /<b/) ||
         ($line =~ /<t/) ||
         ($line =~ /<i/) ||
         ($line =~ /href/) ||
         ($line =~ /<form/) ||
        ($line =~ /<\!--/) ||
         ($line =~ /<a/))
          {
      # note: this is pretty simplistic: actually, I think real html doesn't
      # care if all the parts of the tag are on the same line, so I think
      # <ht  on one line and  ml> on the next would be a valid html tag, and
      # would fool us here because we aren't keeping track of angle brackets
      # across line boundaries.  if the spammers start getting tricky, then
      # we'll need to get more sophisticated here
      if ( ($line =~/\@/) && !($line =~/mailto/)) {
       #do nothing, it's probably just an email address
      } elsif ( (!($line =~/to:/))
      && (!($line =~/from:/))
      && (!($line =~/return-path:/))
      && (!($line =~/message-Id:/))
      && (!($line =~/sender:/)) )
      { #it's not a header line
       $done =1;
       $is_html=1;
      }
     } elsif ($count>19) { # if it isn't in the first 20 lines, assume text
      $done=1;
     }
   }
        my $finger_set;
        if ($is_html>0) { #the message is html
         $finger_set = procTextHtml($entity);
        } else { #the message is plain text
         $finger_set = procTextPlain($entity);
        }
        return $finger_set
}
sub get_version{
 return $version;
}
1; # end of MessageFactory.pm
```

```
# created 11-10-02 by Liz Derr
# These tables belong to a separate database that uninspected
# messages are stored in while they're waiting in the queue
# non-spam messages are also stored in this database.
# this is a separate database from accessio_handprints because
# this data is solely for use at HQ, and does not pertain to
# the handprint data that gets sent to the filters.
#
USE queued_messages;
DROP TABLE IF EXISTS Category;
CREATE TABLE Category (
Category_id  smallint unsigned primary key not null,
Category_name  varchar(20) not null,
Category_desc  varchar(60),
Category_lastmod_date timestamp,
Unique U_Category_name(Category_name)
);
#
# These constants must be the same as those in the WWW::Constants.pm
#
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("1","Unreviewed",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("2","Adult",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("3","Money Making",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("4","Free Stuff",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("5","Gaming/Casino",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("6","Stuff For Sale",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("7","Virus",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("8","Other",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("9","Not Sure",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("10","Unclassified Spam",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("11","Not Spam",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("12","Loans",NOW());
INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("13","Prescriptions",NOW());
 INSERT INTO Category (Category_id,Category_name,Category_lastmod_date)
  VALUES("99","Problem",NOW());
##################################################################
#create tables for Fingers
```

```
DROP TABLE IF EXISTS Finger;
CREATE TABLE Finger (
Finger_id  int unsigned primary key auto_increment not null,
Finger_Finger_name_id int not null,
Finger_Message_id int not null references Message(Message_id),
Finger_Finger_set_id int not null references Finger_set(Finger_set_id),
Finger_digest_code char(22) not null,
Finger_weight  decimal(6,4) not null default 1,
Finger_length  int unsigned,
Finger_inserted_on datetime,
Finger_updated_on timestamp,
Index   I_Finger_digest(Finger_digest_code),
Index   I_Finger_setid(Finger_Finger_set_id),
Index   I_Finger_msgid(Finger_Message_id)
);
DROP TABLE IF EXISTS Finger_set;
CREATE TABLE Finger_set (
Finger_set_id  int unsigned primary key auto_increment not null,
Finger_set_Message_id int not null references Message(Message_id),
Finger_set_parentset_id int not null references Finger_set(Finger_set_id),
Finger_set_tree_index int,
Finger_set_inserted_on datetime,
Finger_set_updated_on timestamp,
Index I_Finger_set_msgid(Finger_set_Message_id),
Index I_Finger_set_parentsetid(Finger_set_parentset_id)
);

####################################################################
#create tables for Message
DROP TABLE IF EXISTS Message;
CREATE TABLE Message (
Message_id  int unsigned primary key auto_increment not null,
Message_Category_id smallint unsigned references Category(Category_id),
Message_filename char(32) not null,
Message_pathname varchar(64) not null,
Message_inserted_on datetime,
Message_updated_on timestamp,
Unique  U_Message_filename(Message_filename),
Index   I_Message_category(Message_Category_id)
);
DROP TABLE IF EXISTS QueueHash;
create table QueueHash (
InternalID bigint(21) NOT NULL auto_increment,
MD5SUM varchar(64) NOT NULL,
MessageId int,
SeenCount int,
CategoryId int,
Inserted_on datetime,
Updated_on timestamp,
Primary Key(InternalID),
Index   I_Hash_MD5Sum(MD5SUM),
Index   I_Hash_Message_id(MessageId),
Index   I_Hash_SeenCount(SeenCount),
Index   I_Hash_CategoryId(CategoryId),
Index   I_Hash_Inserted_on(Inserted_on)

);
```

```
#  Copyright Notice:    Miavia, Inc
#                      Copyright 2002,2003,2004
#                      All Rights Reserved
#  $Id: NewHtmlBodyProcessor.pm,v 1.72 2004/06/04 07:29:06 lizd Exp $
#
#
# modified 9-24-02 by Liz Derr - added support for multiple fingers, this now obsoletes
#                              HtmlBodyProcessor
#                              note: need to add support for html paragraphs
# 2-15-04 clear of Settings references

package MiaVia::NewHtmlBodyProcessor;
## An HtmBodyProcessor extracts Fingers from an entity body of type
## text/html, for use in constructing the message's Finger objects
use strict;
use HTML::Parser;
our @ISA = qw( HTML::Parser );
use IO::String;
use MiaVia::NewTextBodyProcessor;
use MiaVia::HtmlTagProcessor;
use MiaVia::Utils;
## CLASS METHODS
##
# create a new HtmlBodyProcessor
#
sub new {
  my $class = shift;
  my $self = HTML::Parser->new( api_version => 3
                              , start_h => [\&start_tag,
                                           "self, tagname, attr,attrseq"]
                              , end_h   => [\&end_tag,
                                           "self, tagname"]
                              , text_h  => [\&text,
                                           "self, dtext, is_cdata"]
                              );

  my $textProcessor = MiaVia::NewTextBodyProcessor->new(1);
  $self->{textProcessor} = $textProcessor;

  my $htmlTagProcessor = MiaVia::HtmlTagProcessor->new();
  $self->{htmlTagProcessor} = $htmlTagProcessor;
  $self->{scripting} = 0; # not in the middle of a script tag (yet)
  $self->{nested} = ""; # no nested HTML text (yet)

  bless $self, $class;
}
## INSTANCE METHODS
##
# add text to be processed for the body
#
```

```
sub add {
  my ($self, $text) = @_;
  $self->parse($text);
}
sub getNested {
  my ($self) = @_;
  return $self->{nested};
}
sub resetNested {
  my ($self) = @_;
  $self->{nested} = "";
}
# notify that all text has been "added"
#
sub done {
  my $self = shift;
  $self->eof();
}
# get stripped body text
#
sub stripped_body {
  my $self = shift;
  my $textProcessor = $self->{textProcessor};
  return $textProcessor->stripped_body();
}
# get stripped body text
#
sub paragraphs {
  my $self = shift;
  my $textProcessor = $self->{textProcessor};
  return $textProcessor->paragraph_list();
}
sub hrefs {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->href_list();
}
sub linksubs {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->linksub_list();
}
sub mailtos {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->mailto_list();
}
sub images {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->image_list();
}
## PARSER METHODS ("CALLBACKS")
##
# action to take on a start tag
#
sub start_tag {
   my ($self, $tagName, $attr, $attrseq) = @_;

    if($tagName eq "body") {
      my $textProcessor = $self->{textProcessor};
      $textProcessor->activate();

    } elsif ($tagName eq "head") {
      my $textProcessor = $self->{textProcessor};
      $textProcessor->deactivate();
      $textProcessor->reset();
    } elsif (($tagName eq "a") || ($tagName eq "link")) {
        my %atthash = %{$attr};
           my @attlist = @{$attrseq};
        foreach my $attribute (@attlist)  {
         #print "a or link attribute is $attribute\n";
         my $htmlTagProcessor = $self->{htmlTagProcessor};
         $htmlTagProcessor->add($attribute,$atthash{$attribute});
      }
    } elsif (($tagName eq "img") || ($tagName eq "area") || ($tagName eq "iframe") ||($tagName eq "map")) {
        my %atthash = %{$attr};
           my @attlist = @{$attrseq};
        foreach my $attribute (@attlist)  {
         my $htmlTagProcessor = $self->{htmlTagProcessor};
         $htmlTagProcessor->add($attribute,$atthash{$attribute});
        }
     } elsif ($tagName eq "script") {
       my %atthash = %{$attr};
       my @attlist = @{$attrseq};
       $self->{scripting} = 1; # we're now in a script tag
       foreach my $attribute (@attlist)  {
        my $decoded = MiaVia::Utils::decode_hex($attribute);
      }

     } elsif ($tagName eq "form") {
       my %atthash = %{$attr};
       my @attlist = @{$attrseq};
       foreach my $attribute (@attlist)  {
        my $htmlTagProcessor = $self->{htmlTagProcessor};
         $htmlTagProcessor->add($attribute,$atthash{$attribute});
      }

     } else {
      #print "unknown tag $tagName\n";
     }
}
# action to take on an end tag
#
sub end_tag {
  my ($self, $tagName) = @_;
  if ($tagName eq "head") {
    my $textProcessor = $self->{textProcessor};
    $textProcessor->activate();
  } elsif ($tagName eq "script") {
       $self->{scripting} = 0; # no longer in a script tag
  }
}
# action to take on text
#
sub text {
  my ($self, $text, $is_cdata) = @_;

  if ($self->{scripting}) { # we're in a script tag

   my $line = $text;
   $line =~ tr/A-Z/a-z/;
  if (($line =~ /<html>/) ||
         ($line =~ /text\/html/) ||
         ($line =~ /<p/) ||
         ($line =~ /<b/) ||
         ($line =~ /<t/) ||
         ($line =~ /<i/) ||
         ($line =~ /href/) ||
         ($line =~ /<form/) ||
        ($line =~ /<\!--/) ||
         ($line =~ /<a/)) {
           $self->{nested} = $text;
          }

  } elsif (length($text) > 0) {
 my $textProcessor = $self->{textProcessor};
 $textProcessor->add($text);
 $textProcessor->HtmlParagraphMaker($text);
  } # end if length > 0

} # end sub
1; # end of NewHtmlBodyProcessor.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: NewScore.pm,v 1.20 2004/03/23 03:17:22 lizd Exp $
#
# modified 9-24-02 by Liz Derr - added support for multiple fingers, this now
#                               obsoletes Score.pm
# 2-19-04 added category
# 2-21-04 added userMatch and cacheScore for Joe Harris

package MiaVia::NewScore;
use MiaVia::SysLogEvent qw(debug_msg);
use MiaVia::DbUtil;
## A Score object represents an email's SPAM score.
use strict;
## CLASS METHODS
##
# Create a Score object, given
#
sub new {
  my ($class, $message, $match, $userMatch, $cacheScore, $dbh) = @_;

  # for backward compatibility
  if (!(defined $userMatch)) {
   $userMatch = 0;
  }
  if (!(defined $cacheScore)) {
   $cacheScore = 0;
  }
  my $self = { score => 0
          , matchId => "none"
          , match_count => 0
          , category => 1 # category of 1 = Unreviewed
             };
  my $maxScore = 0;
  my $count = 0;
  my @matchList = @{$match->matchList()};
  foreach my $elem (@matchList) {
   $count++;
 if ( $elem->{score} >= $maxScore) {
    $self->{score} = $elem->{score};
    $maxScore = $elem->{score};
    $self->{matchId} = $elem->{message_id};
    #$self->{category} = $elem->{category};
 }
   }

   my $match_category = MiaVia::DbUtil::getCategory($dbh, $self->{matchId});
   if ($match_category) {
    $self->{category} = $match_category;
   }
   $self->{match_count} = $count;

  bless $self, $class;
}
## INSTANCE METHODS
##
# Getters
sub getMatchId {
  my $self = shift;
  if (exists $self->{matchId}) {
    return $self->{matchId};
  } else {
   debug_msg("match id does not exist");
  }
}
sub getScore {
  my $self = shift;
  return $self->{score};
}
sub getCount {
  my $self = shift;
  return $self->{match_count};
}
sub getCategory {
  my $self = shift;
  return $self->{category};
}
sub setCategory {
  my $self = shift;
  my $category = shift;
  $self->{category} = $category;
}
1; # end of NewScore.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: PipelineUtils.pm,v 1.34 2004/05/20 08:41:28 lizd Exp $
#
#
# these are general utilities
#
# created 8-07-02 by Liz Derr
# functions added 6-09-03 by Erik Kargard
# Removed References to MiaVia::Settings (mostly debugs), and obsolete get_dbversion,
# also removed use DbUtil;
# created PipelineUtils on 2-15-04 with pipeline-only routines
#############PIPELINE###############
package MiaVia::PipelineUtils;
use MiaVia::Message;
use MiaVia::NewFingerSet;
use MIME::Parser;
use DBI;
my %dbPasswd = (
                 "miavia_rdr" => "accessio"
               , "miavia_wrtr" => "panther"
               );
#
#
sub look_for_forwards {

     my $entity = shift;
     my $remove = 'N'; #indicates whether there are forwards to remove
     my $recip = $entity->get('From');
     my $to = $entity->get('To');
     my $contype = $entity->get('Content-Type');
     my $forwarded_inline = $entity->stringify_body;
     if ($contype ) {
       if (($contype =~ /mixed/) ||
        ($contype =~ /digest/) ||
        ($contype =~ /related/)) {
        $remove = 'N';
       } elsif  ($to && $recip) {
          if (($to eq "") and ($recip eq "")) {
             $remove = 'N';
          } elsif (($to =~ /Edmonson/) || ($recip =~ /Edmonson/)) {
             $remove = 'Y';
          }
       } elsif  # if included message
       (($forwarded_inline =~ /-----Original Message-----/)||
       ($forwarded_inline =~ /----- Original Message -----/)||
       ($forwarded_inline =~ /> ----------/) ||
       ($forwarded_inline =~ /Received: from/) ||
       ($forwarded_inline =~ /Auto forwarded by a Rule/) ||
      ($forwarded_inline =~ /-----Forwarded Message-----/)) {
       $remove = 'Y';
     } # end elsif
    }# end if contype
return $remove;
}
#
# this routine attempts to remove the embedded headers in
# email messages that are forwarded "in line"
# It doesn't work perfectly, and might alter an email
# message in damaging ways by deleting more than the headers
# it is also specific to the formats that some of our regular
# contributors forward messages in (i.e. Edmonson)
#
# NOTE: it doesn't deal explicitly with multi-part or encoded
# messages
#
# BUG: it sometimes eats HTML overzealously in "funky" and
# I don't know why; also, in "standard", it will eat only the
# first line of the second instance of a forwarded header in
# a message, and I don't know why
#
# returns the "scrubbed" stringified body
#
# always save a copy of the original message
# 8-13-02 by Liz Derr
#
sub remove_forwards { # returns the cleaned up string
 my $body = shift; # assumes body is stringified
 my $recip = shift;
 my @lines;
 my @scrubbed;
 my $done = 0;
 my $line;
 my $funky = 0;
 my $standard = 0;
 my $hungry = 0;

  @lines = split (/\n/,$body);
  if ($recip =~ /Edmonson/) {$funky =1;}
  foreach $line (@lines) {
    if ($done eq 0) { # look for stuff to remove
     if (($line =~ /-----Original Message-----/) ||
       ($line =~ />-----Original Message-----</) ||
            ($line =~ /-------- Original Message --------/)||
            ($line =~ /----- Original Message -----/)||
            ($line =~ /-----Forwarded Message-----/)) {
           #set standard on to chop remaining header lines
           # print "standard: $line\n";
           $standard = 1;
  $hungry = 1;
 } elsif  ( (($standard eq 1) and
          ($hungry eq 1)) and
((($line =~ /From: /) ||
($line =~ /To: /) ||
  ($line =~ /Subject: /) ||
 ($line =~ /Sent: /) ||
 ($line =~ /Cc: /) ||
 ($line =~ /Date: /) ||
 ($line =~ /Reply-To: /) ||
 ($line =~ /Delivered-To: /) ||
 ($line =~ /Return-Path: /) ||
 ($line =~ /X-ClientAddr: /) ||
 ($line =~ /Message-Id: /) ||
 ($line =~ /Content-Type: /) ||
 ($line =~ /^,/) ||
 ($line =~ /CC:/))) {
  # skip the line
 # print "stand skip: $line\n";
} elsif  (($standard eq 1) and ($hungry eq 1) and
       ($line =~ /^\s$/)) { # stop eating lines at whitespace
        # print "standard: not hungry at whitespace\n";
        $hungry = 0;
 } elsif  (( $funky eq 1) and
       ($line =~/>Auto for/)) { # stop eating html lines after "Rule"
       # print "funky: stopped at html Rule\n";
        $hungry = 0;
        $done = 1;
 } elsif  (( $funky eq 1) and
       ($line =~/^> Auto forwarded by a Rule/)) { # stop eating text lines after "Rule"
        # print "funky: not hungry at text Rule\n";
        $hungry = 0; # but not done yet, look for html part
 } elsif  (( $funky eq 1) and
       ($hungry eq 1)) { # eat lines while hungry
       # print "HUNGRY: $line\n";
 } elsif  (( $funky eq 1) and
  ($line =~ /\"Arial\">----------</)) {   # funky html
    # print "funky: $line\n";
    $hungry = 1; #start eating  lines
    # skip the line
 } elsif  (( $funky eq 1) and
  ($line =~ /^> ----------/)) {   #funky text, not html
    # print "funky: $line\n";
    $hungry = 1; #start eating  lines
    # skip the line
 } else { # not done, but no bad lines, so
     # push the line into the scrubbed list
push @scrubbed, "$line\n";
}
} else { # end if not done
```

```
        push @scrubbed, "$line\n";
      }# end else
   } # end foreach
return @scrubbed;
}
sub markAllFingersWeight {
  my ($dbh, $digest_code, $weight) = @_;

  $updateStr = qq { UPDATE Finger SET Finger_weight='$weight'
                      WHERE Finger_digest_code='$digest_code'
                   };
  $sth = $dbh->prepare($updateStr);
  $result = $sth->execute();
  $sth->finish();

  return $result;
}
sub deleteAllFingers {
  my ($dbh, $digest_code) = @_;

  $updateStr = qq { delete from Finger WHERE Finger_digest_code='$digest_code'};
  $sth = $dbh->prepare($updateStr);
  $result = $sth->execute();
  $sth->finish();

  return $result;
}
# This is how we will treat "noise" fingers, and fingers with text that
# is not indicative of spam and should therefor not be considered in a
# match decision
# sets the finger type to Irrelevant (11), and the Finger weight to 0.001
sub markAllFingersAsIrrelevant {
  my ($dbh, $digest) = @_;

  my $updateStr = qq { UPDATE Finger SET Finger_Finger_name_id='11', Finger_weight='0.001'
                      WHERE Finger_digest_code='$digest'
                   };
  my $sth = $dbh->prepare($updateStr);
  my $result = $sth->execute();
  $sth->finish();

  return $result;
}
# given a message id, get the category of the message from the database
sub getWeight {
 my ($dbh, $digest) = @_;

   my $weight = 1;
 my $queryStr = qq{ select Finger_weight FROM Finger
  WHERE Finger_digest_code='$digest' LIMIT 1};
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 ($weight) = $sth->fetchrow_array();
 $sth->finish();

  return $weight;
}
# given a message id, get the category of the message from the database
sub getSpecificWeight {
 my ($dbh, $digest, $msgid) = @_;

   my $weight = 1;
 my $queryStr = qq{ select Finger_weight FROM Finger
  WHERE Finger_digest_code='$digest' and Finger_Message_id='$msgid'};

 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 ($weight) = $sth->fetchrow_array();
 $sth->finish();

  return $weight;
}
# given a message id, get the category of the message from the database
sub setCategory {
 my ($dbh, $msg_id, $cat_type) = @_;

 my $queryStr = qq{ UPDATE Message SET Message_Category_id='$cat_type'
  WHERE Message_id='$msg_id'};

 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();
 $sth->finish();

  return $ret;
}
sub get_fingers {
 my ($class, $fingerSet) = @_;

 my @finger_list;
 my @sub_fingers;

 foreach my $subFingerSet (@{$fingerSet->fingerSets()}) {
      @sub_fingers = get_fingers($self, $subFingerSet);
      foreach my $subfing (@sub_fingers) {
       push @finger_list, $subfing;
      }
 }
 #warn "get_fingers after fingerset";
 foreach my $finger (@{$fingerSet->fingers()}) {
    #my $finger_string = $finger->xmlString();
    push @finger_list, $finger;
    #print "$finger_string\n";
    #my $tmp_digest = $finger->digest();
      #print "checking finger with digest $tmp_digest\n";
      #if ($tmp_digest eq $digest_code) {
      #   print "value of finger: $finger->value\n";
      #}
  }
  #warn "get_fingers after finger loop";
  return @finger_list;
}
sub getMessageFilenames {
 my ($dbh, $digest) = @_;

   my @ids;
   my @names;

 my $queryStr = qq{ SELECT Finger_Message_id
  FROM Finger WHERE Finger_digest_code='$digest'};

 my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 while (my ($id) = $sth->fetchrow_array()) {
    push @ids, $id;
 }

 if ( (scalar(@ids)) > 0) {

  my $queryStr = qq{ SELECT Message_filename
   FROM Message WHERE Message_id IN };
  my $sub_string = shift @ids;
  $queryStr .= "(\"$sub_string\"";
  foreach $sub_string (@ids) {
        $queryStr .= ",\"$sub_string\"";
  }
  $queryStr .= ")";

  my $sth = $dbh->prepare($queryStr);
  $sth->execute();
  while (my ($name) = $sth->fetchrow_array()) {
     push @names, $name;
  }
 }

 $sth->finish();

  return @names;
}
```

```
sub insertCommonFinger {
 my $dbh = shift;
 my $digest = shift;
 my $value = shift;

 my $insertStr = qq{ INSERT INTO CommonFinger (CommonFinger_digest_code, CommonFinger_value,
CommonFinger_inserted_on)
  VALUES ("$digest","$value",'NOW()') };
 my $sth = $dbh->prepare($insertStr);
 $sth->execute();
 $sth->finish();
}
# added 11-21-03 to support inspection across networked machines
sub openNetworkQueueDb {
  my $user = shift;
  my $host = shift;
  my $dbh = DBI->connect("DBI:mysql:queued_messages:$host"
                       , $user
                       , $dbPasswd{$user}
                       , { RaiseError => 1 });
  return $dbh;
}
sub openQueueDb {
  my $user = shift;
  my $dbh = DBI->connect("DBI:mysql:queued_messages:localhost"
                       , $user
                       , $dbPasswd{$user}
                       , { RaiseError => 1 });
  return $dbh;
}
# is Queued returns the SeenCount for the Queue Message Id.  Returns 0 if not queued.
sub isQueued {
  my $msgid = shift;
  my $dbh = shift;
  my $seen = 0;
  my $queryStr = qq{select SeenCount FROM QueueHash WHERE MessageId =};
  $queryStr .= "\"$msgid\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my $numRows = $sth->rows;
  if( ($sth->rows) > 0) {
 ($seen) = $sth->fetchrow_array();
  }
  return $seen;
}
# is Queued returns the SeenCount for the Queue Message Id.  Returns 0 if not queued.
sub getSeenByName {
  my $fname = shift;
  my $dbh = shift;
  my $seen = 0;
  my $queryStr = qq{select SeenCount FROM QueueHash WHERE MD5SUM =};
  $queryStr .= "\"$fname\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  ($seen) = $sth->fetchrow_array();
  return $seen;
}
# is Queued returns the SeenCount for the Queue Message Id.  Returns 0 if not queued.
sub getMostSeen {
  my $dbh = shift;
  my $fname = "";
  my $queryStr = qq{select MD5SUM FROM QueueHash WHERE CategoryId='1'
    order by SeenCount DESC, Updated_on DESC LIMIT 1};
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  ($fname) = $sth->fetchrow_array();
  return $fname;
}
# is Queued returns the SeenCount for the Queue Message Id.  Returns 0 if not queued.
sub updateQueueSeenCount {
  my $msgid = shift;
  my $newcount = shift;
  my $dbh = shift;
  my $queryStr = qq{update QueueHash set SeenCount='$newcount', Updated_on=NOW() WHERE MessageId =};
  $queryStr .= "\"$msgid\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();

}
sub HashIdFromName {
  my $md5sum = shift;
  my $dbh = shift;
  my $cacheId = 0;
  my $queryStr = qq{select MessageId FROM QueueHash WHERE MD5SUM =};
  $queryStr .= "\"$md5sum\"";
  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my $numRows = $sth->rows;
  if( ($sth->rows) > 0) {
 ($cacheId) = $sth->fetchrow_array();
  }
  return $cacheId;
}
sub insertQueueHash {
 my $md5sum = shift;
 my $queue_msgid = shift;
 my $dbh = shift;
 my $insertStr = qq{ INSERT INTO QueueHash (MessageID, MD5SUM, SeenCount, CategoryId, Inserted_on,
Updated_on)
```

```
  VALUES ("$queue_msgid","$md5sum","1","1", NOW(), NOW()) };
 my $sth = $dbh->prepare($insertStr);
 $sth->execute();
 $sth->finish();
}
sub deleteQueueHash {
 my $queue_msgid = shift;
 my $dbh = shift;
 my $deleteStr = qq{ delete from  QueueHash WHERE MessageId =};
   $deleteStr .= "\"$msgid\"";
 my $sth = $dbh->prepare($deleteStr);
 $sth->execute();
 $sth->finish();
}
sub deleteQueueHashMD5 {
 my $MD5 = shift;
 my $dbh = shift;
 my $deleteStr = qq{ delete from QueueHash WHERE MD5SUM =};
   $deleteStr .= "\"$MD5\"";
 my $sth = $dbh->prepare($deleteStr);
 $sth->execute();
 $sth->finish();
}
sub updateMessageCategory {
 my $msgid = shift;
 my $catid = shift;
 my $dbh = shift;
   my $queryStr = qq{update Message set Message_Category_id='$catid' WHERE MessageId =};
 $queryStr .= "\"$msgid\"";
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();

 return $ret;
}
sub updateQueueHashCategory {
 my $fname = shift;
 my $catid = shift;
 my $dbh = shift;
   my $queryStr = qq{update QueueHash set CategoryId='$catid', Updated_on=NOW() WHERE MD5SUM =};
 $queryStr .= "\"$fname\"";
 my $sth = $dbh->prepare($queryStr);
 my $ret = $sth->execute();

 return $ret;
}
sub deleteWildLink {
  my $dbh = shift;
  my $msg_id = shift;
  my $wild_string = shift;
  # get wild links id
  my $queryStr = qq{select WildLink_id FROM WildLink WHERE
   WildLink_Message_id='$msg_id' and WildLink_link_string='$wild_string'};

  my $sth = $dbh->prepare($queryStr);
  my $ret = $sth->execute();
  my ($wild_id) = $sth->fetchrow_array();


  if ($wild_id) {
    my @sublink_ids;

   # get sublink ids for those wild links
   $queryStr = qq{ SELECT SubLink_id
   FROM SubLink WHERE SubLink_WildLink_id='$wild_id'};
   $sth = $dbh->prepare($queryStr);
   $ret = $sth->execute();
   while (my ($sublink_id) = $sth->fetchrow_array()) {
     push @sublink_ids, $sublink_id;
   }

   # remove sub link entries
   $deleteStr = qq{delete FROM SubLink WHERE SubLink_id IN };
   $sub_string = shift @sublink_ids;
   $deleteStr .= "(\"$sub_string\"";
   foreach $sub_string (@sublink_ids) {
        $deleteStr .= ",\"$sub_string\"";
   }
   $deleteStr .= ")";
   $sth = $dbh->prepare($deleteStr);
   $ret = $sth->execute();


   # remove wild links
   $deleteStr = qq{delete FROM WildLink WHERE WildLink_id='$wild_id'};
   $sth = $dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $sth->finish();
  } # end if wild link id
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: FingerSpec.pm,v 1.5 2004/02/16 04:03:54 lizd Exp $
# 2-15-04 clear of Settings references
#
package MiaVia::FingerSpec;
## FingerSpec specifies how to extract Fingers from specific
## types of MIME entities.
## CLASS METHODS
##
sub new {
  my ($class, $spec) = @_;
  my $self = { spec => $spec };
  bless $self, $class;
}
## INSTANCE METHODS
##
# does the FingerSpec have a method for this type?
#
sub hasMethod {
  my ($self, $type) = @_;
  exists $self->{spec}->{$type}
}
# get the method for this type
#
sub getMethod {
  my ($self, $type) = @_;
  return $self->{spec}->{$type}
}
1; # end of FingerSpec.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: NewFinger.pm,v 1.24 2004/06/04 02:31:54 lizd Exp $
#
# modified 9-24-02 by Liz Derr to support multiple fingers
# modified 10-29-02 by Liz Derr to support non-text fingers, minimal
#                               stripping, and html tag fingers
#
# 2-15-04 clear of Settings references
package MiaVia::NewFinger;
# A Finger is one characteristic of an email message. The Finger object
# may contain a string value extracted from the message and the strings's
# digest or just the digest.
#
# Each Finger has a name and version, which precisely identifies the
# code used to extract the finger's string from the email. The finger
# also has a type ID, which is the ID in the finger_name table of this
# finger's name/version.
# 2-26-04 added finger type of irrelevant
use strict;
use Digest::MD5;
#use XML::LibXML;
## PACKAGE DATA
##
our %fingerType = ( 'stripped_body' =>
                     {
                       '1' => '1' #version is 1, type_id is 1
                      , '2' => '1' #version is 2, type_id is 1
                     },
                   'stripped_paragraph' =>
                     {
                       '1' => '2' #version is 1, type_id is 2
                      , '2' => '2' #version is 2, type_id is 2
                     },
                    'min_stripped_paragraph' => # added 10-29-02
                     {
                       '1' => '3' #version is 1, type_id is 3
                     },
                    'href_tag' => # added 10-29-02
                     {
                       '1' => '4' #version is 1, type_id is 4
                     },
                    'mailto_tag' => # added 10-29-02
                     {
                       '1' => '5' #version is 1, type_id is 5
                     } ,
                     'image_tag' => # added 11-21-02
                     {
```

```
                    '1' => '6' #version is 1, type_id is 6
                  } ,
                 'non_text' => # added 10-29-02
                  {
                    '1' => '7' #version is 1, type_id is 7
                  },
                  'min_stripped_body' => # added 10-29-02
                  {
                    '1' => '8' #version is 1, type_id is 8
                  },
                  'link_sub' => # added 11-19-03
                  {
                    '1' => '9' #version is 1, type_id is 9
                  },
                  'common' => # added 11-19-03
                  {
                    '1' => '10' #version is 1, type_id is 10
                  },
                  'irrelevant' => # added 2-26-04
                  {
                    '1' => '11' #version is 1, type_id is 11
                  },
                );
## CLASS METHODS
##
# construct a new Finger object
#
sub new {
  my ($class,       # 'Finger' or sub-class
      $name,        # finger name, required
      $version,     # finger version, required
      $optArgsRef,  # hash of optional args (value, digest)
     ) = @_;
  my $self = { name => $name
           , version => $version
           , type_id => $fingerType{$name}->{$version}
           ,weight => 1
           };

  if ($optArgsRef) {
    my %optArgs = %{$optArgsRef};
    if (defined $optArgs{value}) {
      $self->{value} = $optArgs{value};
      $self->{finglen} = length($optArgs{value});
    }
    if (exists $optArgs{digest}) {
      $self->{digest} = $optArgs{digest};
    } elsif (defined $self->{value}) {
        my $ctx = Digest::MD5->new;
        $ctx->add($self->{value});
        $self->{digest} = $ctx->b64digest;
    }
  }

  bless $self, $class;
}
## INSTANCE METHODS
##
# redigest, that is, recompute the digest if
# we have the string value
sub redigest {
  my $self = shift;
  if (defined $self->{value}) {
    my $ctx = Digest::MD5->new;
    $ctx->add($self->{digest});
    $self->{digest} = $ctx->b64digest;
  }
}
# XML METHODS
# create an XML element representing the Finger
#
#sub xmlElem {
  #my $self = shift;
  #my $fingerElem = XML::LibXML::Element->new('Finger');
  # attributes
  #$fingerElem->setAttribute('name', $self->{name});
  #$fingerElem->setAttribute('version', $self->{version});
  #$fingerElem->setAttribute('type_id', $self->{type_id});
  # value sub-element
  #if (defined $self->value) {
    #my $valueElem = XML::LibXML::Element->new('value');
    #$fingerElem->appendChild($valueElem);
    # text
    #my $valueElemText = XML::LibXML::Text->new($self->{value});
    #$valueElem->appendChild($valueElemText);
  #}
  # digest sub-element
  #if (defined $self->digest) {
   # my $digestElem = XML::LibXML::Element->new('digest');
   # $fingerElem->appendChild($digestElem);
    # text
    #my $digestElemText = XML::LibXML::Text->new($self->{digest});
   # $digestElem->appendChild($digestElemText);
#  }
#  return $fingerElem;
#}
# render the XML object as a String
#
#sub xmlString {
 # my $self = shift;
 # return $self->xmlElem->toString();
#}
# GETTERS
# getter for 'name'
#
sub name {
  my $self = shift;
  return $self->{name};
}
# getter for 'version'
#
sub version {
  my $self = shift;
  return $self->{version};
}
# getter for 'type_id'
#
sub type_id {
  my $self = shift;
  return $self->{type_id};
}
# getter for 'value'
#
sub value {
  my $self = shift;
  return $self->{value};
}
# getter for 'digest'
#
sub digest {
  my $self = shift;
  return $self->{digest};
}
# getter for 'finglen' (finger length)
#
sub finglen {
  my $self = shift;
  return $self->{finglen};
}
sub weight {
  my $self = shift;
  return $self->{weight};
}
1; # end of Finger.pm
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
# created 11-21-03 by Liz Derr
# add these tables as part of the accessio_handprints database
USE accessio_handprints;
##################################################################
#create tables for WildLinks - these are link strings with wildcards
DROP TABLE IF EXISTS WildLink;
CREATE TABLE WildLink (
WildLink_id  int unsigned primary key auto_increment not null,
WildLink_Message_id int not null references Message(Message_id),
WildLink_Finger_set_id int not null references Finger_set(Finger_set_id),
WildLink_link_string varchar(220) not null,
WildLink_weight  decimal,
WildLink_inserted_on datetime,
WildLink_updated_on timestamp,
Index   I_WildLink_link_string(WildLink_link_string),
Index   I_WildLink_Finger_set_id(WildLink_Finger_set_id),
Index   I_WildLink_Message_id(WildLink_Message_id)
);
# these are components of link strings, used for lookups into WildLink
DROP TABLE IF EXISTS SubLink;
CREATE TABLE SubLink (
SubLink_id  int unsigned primary key auto_increment not null,
SubLink_WildLink_id int not null references WildLink(WildLink_id),
SubLink_sub_string varchar(50) not null,
SubLink_type  varchar(10),
SubLink_inserted_on datetime,
SubLink_updated_on timestamp,
Index I_SubLink_WildLink_id(SubLink_WildLink_id),
Index I_SubLink_sub_string(SubLink_sub_string)
);
# has fingers that should be ignored when doing lookups - these are MD5 hex
DROP TABLE IF EXISTS CommonFinger;
CREATE TABLE CommonFinger (
CommonFinger_id   int unsigned primary key auto_increment not null,
CommonFinger_Finger_name_id int not null references Finger_name(Finger_name_id),
CommonFinger_digest_code char(22) not null,
CommonFinger_value  varchar(255),
CommonFinger_inserted_on datetime,
CommonFinger_updated_on  timestamp,
Index   I_CommonFinger_digest_code(CommonFinger_digest_code)
);
# has sub links that should be ignored when doing lookups -these are text
DROP TABLE IF EXISTS CommonSubLink;
CREATE TABLE CommonSubLink (
CommonSubLink_id  int unsigned primary key auto_increment not null,
CommonSubLink_Finger_name_id int not null references Finger_name(Finger_name_id),
CommonSubLink_sub_string varchar(50) not null,
CommonSubLink_inserted_on datetime,
CommonSubLink_updated_on timestamp,
Index   I_CommonSubLink_digest_code(CommonSubLink_sub_string)
);
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#  $Id: NewMimeProcessor.pm,v 1.57 2004/05/27 02:45:13 lizd Exp $
#
package MiaVia::NewMimeProcessor;
## A MimeProcessor creates a MIME entity from parsing a MIME message.
## It has methods for manipulating and printing that MIME entity.
##
## modified 07-25-02 by Liz Derr - added getHeaderTag method
## modified 09-03-02 by Liz Derr - added call_for_help instead of die
## modified 9-24-02 by Liz Derr - added support for multiple fingers, this now
##                              obsoletes MimeProcessor.pm
## modified 10-29-02 by Liz Derr - added new fingers for attachments
# 2-19-04  added call to MiaVia::Utils to get mime_temp
#use strict;
use MIME::Parser;
use MiaVia::Utils;
## CLASS DATA
##
## CLASS METHODS
##
# construct a new MimeProcessor object
#
sub new {

  my $tmpDir = "/usr/local/accessio/data/tmp/";
  $tmpDir = MiaVia::Utils->get_mime_temp();
  if (!-d $tmpDir) {

   warn "MIME Processor: MIME temp directory does not exist: $tmpDir\n";
  }
  my ($class, $msgSource) = @_;
  my $mimeFile;
  my $srcType = $msgSource->{type};
  my $parser = MIME::Parser->new();
  $parser->output_dir($tmpDir);
  $parser->output_to_core(0);
  $parser->tmp_to_core(0);
  $parser->use_inner_files(0);
  $parser->extract_uuencode(1); # added 1-17-03 by LD
  my $entity;
  if ($srcType eq 'mimeFile') {

   if (defined($msgSource->{dir})) {
    if ($msgSource->{dir} =~ /\/$/) {
        # if the dir already has a trailing slash, don't add one
        $mimeFile = $msgSource->{dir}
                . $msgSource->{fileName} . '.'
                . $msgSource->{fileExtension};
    } else {
        $mimeFile = $msgSource->{dir} . '/'
                  . $msgSource->{fileName} . '.'
                  . $msgSource->{fileExtension};
    }
  } else {
warn "dir for mimeFile not defined\n";
  }
   if (-r $mimeFile) {
      eval {
           $entity = $parser->parse_open($mimeFile)
         };
      if ($@) {
       warn "MIME::Parser error $@ in '$mimeFile'\n";
      }
   }
} elsif ($srcType eq 'mimeString') {
   eval {
        $entity = $parser->parse_data($msgSource->{mimeString});
       };
   if ($@) {
       warn "MIME::Parser error in mimeString\n";
   }
} elsif ($srcType eq 'smtp') {
   if (($msgSource->{dir}) =~/\/$/) {
       $mimeFile = $msgSource->{dir}
                 . $msgSource->{fileName} . '.'
                 . $msgSource->{fileExtension};
   } else {
       $mimeFile = $msgSource->{dir} . "/"
                 . $msgSource->{fileName} . '.'
                 . $msgSource->{fileExtension};
   }
  if (-r $mimeFile) {
      eval {
           $entity = $parser->parse_open($mimeFile)
         };
      if ($@) {
       warn "MIME::Parser error: '$mimeFile'\n";
      }
   } else {
       warn "No such MIME file: '$mimeFile'\n";
   }
} else {
   warn "Cannot create MimeProcessor for MessageSource of type '$srcType'";
}

# calculate the file date in YYYY/DDD format
my $year =0;
my $yday =0;
```

```
  ($year, $yday) = MiaVia::Utils::file_date($mimeFile);
  $yday = $yday--; # for path calculation DDD-1
  my $date = "$year"."/"."$yday";

  if (!(defined($entity))) {
   warn "parser entity is not defined for $mimeFile of type $srcType\n";
  }
  my $self = { parser => $parser
             , entity => $entity
             , fileDate => $date
             #, recipName => $recipName
             };
  bless $self, $class;
}
## INSTANCE METHODS
##
# get the a text field from the header
#
sub getHeaderTag {
  my ($self, $tag) = @_;
  my $head = $self->{entity}->head();
  my $text = $head->get($tag);
  return $text;
}
# add a header, with given $tag and $text
#
sub addHeader {
  my ($self, $tag, $text) = @_;
  my $head = $self->{entity}->head();
  $head->add($tag, $text);
}
# add unique header with given $tag and $text
# delete ALL old occurrences of the tag, first
#
sub addUniqueHeader {
  my ($self, $tag, $text) = @_;
  my $head = $self->{entity}->head();
  $head->replace($tag,$text);
  #$head->delete($tag);
  #$head->add($tag, $text);
}
# prepend $text to every header with tag $tag
#
sub prependToHeader {
  my ($self, $tag, $text) = @_;
  my $head = $self->{entity}->head();
  my @oldHeaders = $head->get_all($tag);
  $head->delete($tag);
  foreach my $oldText (@oldHeaders) {
    my $newText = $text . $oldText;
    $head->add($tag, $newText, -1);
  }
}
# finished with parser
#
sub cleanup {
  my $self = shift;
  $self->{parser}->filer->purge;
}
# print the MIME entity to
#
sub printTo {
  my ($self, $outStream) = @_;
  $self->{entity}->print($outStream);
}
# getter for 'recipName'
#
sub recipName {
  my $self = shift;
  return $self->{recipName};
}
# getter for 'fileDate' in "YYYY/DDD" format
#
sub fileDate {
  my $self = shift;
  return $self->{fileDate};
}
# extract fingers
#
sub extractFingers {
  my ($self, $fingerSpec) = @_;
#  print "extracting fingers\n";
  my $fingerSet=0;
  $fingerSet = extractFingersFromEntity($self->{entity}, $fingerSpec);
  if (!$fingerSet) {

    $fingerSet = MiaVia::NewFingerSet->new();
  }
  return $fingerSet;
}
## SUBROUTINES
##
sub extractFingersFromEntity {
  my ($entity, $fingerSpec) = @_;
  # note: $entity is a NewMimeProcessor (duh!)

if (defined $entity) {
  my $mimeType = $entity->mime_type();
  my ($mimePre,$mimePost) = split "/", $mimeType;
  my $num_parts = $entity->parts;
  my $fingerSet=0;
  if (($mimeType =~ /^multipart/) ||( $mimeType =~ /^message/)) {
    my @fingerSets = ();

    my @parts = $entity->parts();
    foreach my $part (@parts) {
      my $partMimeType = $part->mime_type();
      my $finger = extractFingersFromEntity($part, $fingerSpec);
      push @fingerSets, $finger;
    }
    $fingerSet = MiaVia::NewFingerSet->new(@fingerSets);
    return $fingerSet;
  } elsif ($fingerSpec->hasMethod($mimeType)) {
    $fingerSet = &{$fingerSpec->getMethod($mimeType)}($entity);
    return $fingerSet;
  } elsif ($fingerSpec->hasMethod($mimePre)) {
    $fingerSet = &{$fingerSpec->getMethod($mimePre)}($entity);
    return $fingerSet;
  }
} else {
      return 0;
}
}
1; # end of NewMimeProcessor
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: NewTextBodyProcessor.pm,v 1.38 2004/04/27 07:55:25 lizd Exp $
#
#
# modified 9-24-02 by Liz Derr - added support for multiple fingers, this now obsoletes
#                               TextBodyProcessor.pm
# 2-15-04 clear of Settings references
package MiaVia::NewTextBodyProcessor;
## A TextBodyProcessor extracts Fingers from an entity body of type
## text/plain, for use in constructing a messsage's Finger objects
use strict;
use MiaVia::HtmlTagProcessor;
## CLASS METHODS
##
sub new {
  my ($class, $active) = @_;
  $active = 0 unless $active;

  my $paragraph_in_progress = ""; # for paragraphMaker
  my @paragraphList = ();
  my $self = { state => $active
    , paragraph_in_progress => $paragraph_in_progress
    , paragraphList => \@paragraphList };
   my $htmlTagProcessor = MiaVia::HtmlTagProcessor->new();
   $self->{htmlTagProcessor} = $htmlTagProcessor;

  bless $self, $class;
}
## INSTANCE METHODS
##
# make active (see add method)
#
sub activate {
  my $self = shift;
  $self->{state} = 1;
}
# make inactive (see add method)
#
sub deactivate {
  my $self = shift;
  $self->{state} = 0;
}
# return value of paragraph being constructed
sub getparagraphInProgress {
  my $self = shift;
  return $self->{paragraph_in_progress};
}
# return links in plain text
```

```
sub hrefs {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->href_list();
}
sub linksubs {
  my $self = shift;
  my $htmlTagProcessor = $self->{htmlTagProcessor};
  return $htmlTagProcessor->linksub_list();
}
# clear text buffer
#
sub reset {
  my $self = shift;
  $self->{stripped_body} = "";
  $self->{paragraph_in_progress} = "";
}
# return paragraph list
sub paragraph_list {
  my $self = shift;
  return $self->{paragraphList};
}
# when active, add the new string to the text buffer, but
# ignore if inactive
# 8/28/03 looking for links in text, and making HREF fingers
# of them
#
sub add {
  my ($self, $newStr) = @_;
  my $active = $self->{state};
  return unless $active;

  my $newAddStr;

  # upper-case to lower-case, other white space to space
  $newStr =~ tr/A-Z/a-z/;

  # strip links from text and make link fingers
  if (($newStr =~ /http:/) || ($newStr =~ /www\./)){
   $newAddStr = "";
   my @words = split /\s+/, $newStr;
   foreach my $word (@words) { # split on words since
    if (($word =~ /http:/) || ($word =~ /www\./)) {    # links can't contain spaces
     $self->{htmlTagProcessor}->add('href',$word); #add link finger
    } else { # build non-link text string
     $newAddStr = $newAddStr . $word;
    }
   } # end for each word in the string
  } else { # no links in the text string
   $newAddStr = $newStr;
  }

  # remove "long" words - this was removing too many urls
  #$newAddStr =~ s/(\S{15,})//g;

  # remove words that have both letters and numbers
  # $newAddStr =~ s/(\b[a-zA-Z]\b)//g;
  # remove anything that isn't an alpha character
  $newAddStr =~ s/[^a-zA-Z]//g;
  if ($newAddStr) {
    if (exists $self->{stripped_body}) {
      $self->{stripped_body} = $self->{stripped_body} . $newAddStr;
    } else {
      $self->{stripped_body} = $newAddStr;
    }
  }
}
# return the text buffer
#
sub stripped_body {
  my $self = shift;
  return $self->{stripped_body};
}
sub HtmlParagraphMaker {
my $self = shift;
my $text = shift;
    my $stripped_line ="";

    # upper-case to lower-case, other white space to space
    $text =~ tr/A-Z/a-z/;

    if (($text =~ /http:/)|| ($text =~ /www\./)){
   my @words = split /\s+/, $text;
   foreach my $word (@words) { # split on words since
    #print "HTML examining word |$word|\n";
    if (($word =~ /http:/) || ($word =~ /www\./)) {    # links can't contain spaces
     $self->{htmlTagProcessor}->add('href',$word); #add link finger
     #print "removed link $word from text string\n";
    } else { # build non-link text string
     $stripped_line = $stripped_line . $word;
    }
   } # end for each word in the string
     } else { # no links in the text string
   $stripped_line = $text;
     }

     $stripped_line =~ s/[^a-zA-Z]//g;

     if (length($stripped_line) > 0) {
        # if the line is not blank now, then
        # add the alpha chars to the collector
        $self->{paragraph_in_progress}  = $self->{paragraph_in_progress}.$stripped_line;
 if (length($self->{paragraph_in_progress}) > 10) {
       # if we've collected more than 10 alpha chars, and
      # line contains only white space, then we're done
      # with this paragraph
  push @{$self->{paragraphList}},$self->{paragraph_in_progress};
  $self->{paragraph_in_progress} = ""; #reset
      }
      } # end if length of new line > 0 after stripping
}
##
## SUBROUTINES
##
sub collectParagraph {
# use this when you're processing text one line at a time
# this means you need to keep track of done, and the paragraph
# you're building, so that this routine can keep adding on to
# a particular paragraph until it's done. Then, you have to
# push it onto the paragraph list yourself.
    my $line = shift;
    my $paragraph = shift;
    my $done=0;
    my $stripped_line ="";

  if ($line =~ /http:/) {
   my @words = split /\s+/, $line;
   foreach my $word (@words) { # split on words since
    if ($word =~ /^http:/) {    # links can't contain spaces
    } else { # build non-link text string
     $stripped_line = $stripped_line . $word;
    }
   } # end for each word in the string
  } else { # no links in the text string
   $stripped_line = $line;
  }
  # remove anything that isn't an alpha character
  $stripped_line =~ s/[^a-zA-Z]//g;
  # upper-case to lower-case, other white space to space
  $stripped_line =~ tr/A-Z/a-z/;
     if (length($stripped_line) > 0) {
        # if the line is not blank now, then
        # add the alpha chars to the collector
        $paragraph = $paragraph.$stripped_line;
 if (length($paragraph) > 10) {
       # if we've collected more than 10 alpha chars, and
      # line contains only white space, then we're done
      # with this paragraph
       $done = 1;
      }
    } # end if length of new line > 0 after stripping
return ($paragraph, $done);
}
1; # end of NewTextBodyProcessor.pm
```

```
#!/usr/local/bin/perl
use strict;
use MiaVia::Settings;
use MiaVia::DbUtil;
my $count = 0;
my  $removed_fingers =0;
my  $removed_messages =0;
my  $removed_fingersets =0;
my  $ret =0;
my $deleteStr;
my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
my ($indir) = @ARGV;
#print "indir is $indir\n";
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
#print "indir is $indir\n";
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    #print "opening $indir$fname\n";
    while (my $line = <IN>) {
        if (($line =~ /talk-aboutdogs-rescue\@topica.com/) ||
           ($line =~ /Your Ad on www.collectorcartraderonline.com/) ||
           ($line =~ /webmaster\@ohlone.edu/) ||
           ($line =~ /This is a message from the MailScanner E-Mail Virus Protection Service/) ||
           ($line =~ /wsmith\@wordsmith.org/) ||
           ($line =~ /SDForum Events/) ||
           ($line =~ /A.Word.A.Day/) ||
           ($line =~ /The Daily Reckoning/) ||
           ($line =~ /MWave\@subscribermails.com/) ||
           ($line =~ /blackvault\@topica.com/) ||
           ($line =~ /Sorry, no mailbox here by that name/) ||
           ($line =~ /CHABAD.ORG NEWSLETTER/) ||
           ($line =~ /promed-digest-Owner\@promed.isid.harvard.edu (ProMED Digest)/) ||
           ($line =~ /mailing1\@mail.morningstar.net/) ||
           ($line =~ /oup\@oup.co.uk/) ||
           ($line =~ /listowner\@webber.oup.co.uk /) ||
           ($line =~ /NTBUGTRAQ\@LISTSERV.NTBUGTRAQ.COM/) ||
           ($line =~ /alerter\@my-cast.com/) ||
           ($line =~ /nsb\@ysa.org/) ||
           ($line =~ /ScanMail for Microsoft Exchange has taken action on the message/) ||
           ($line =~ /TammyL\@second-to-none.com/) ||
           ($line =~ /From: Microsoft .NET Passport/) ||
           ($line =~ /content violation/) ||
           ($line =~ /entymology/) ||
           ($line =~ /Verisign/) ||
           ($line =~ /Emerging Infectious Diseases Journal Now Online/) ||
($line =~ /California 4-H Youth Development Office/) ||
($line =~ /the cost of shipping to California/) ||
($line =~ /Project Censored/) ||
($line =~ /chembio-terror\@cnslists.miis.edu/) ||
($line =~ /within five days, some of your messages will be automatically deleted/) ||
($line =~ /www.ancestry.com/) ||
($line =~ /WinProxy and eShield Virus Alert/) ||
($line =~ /Comments and Questions (website form)/) ||
($line =~ /joke-of-the-day/) ||
($line =~ /On-Air and Online at AMC/) ||
($line =~ /Apple enews/) ||
($line =~ /Microsoft Business Solution/) ||
($line =~ /Application Spotlight/) ||
($line =~ /Microsoft Windows eHome Division/) ||
($line =~ /Microsoft FYI Strategy magazine/) ||
($line =~ /Newsletters.Microsoft.com/) ||
($line =~ /Newsletter\@HealthandWellnessClub.com/) ||
($line =~ /newsletters\@business2.com/) ||
($line =~ /The PCWorld.com newsletters may contain/) ||
($line =~ /Sondernewsletter /) ||
($line =~ /The FTPplanet.com newsletter is a service/) ||
($line =~ /economist-newsletters-admin\@news.economist.com/) ||
($line =~ /www.earthchangestv.com/) ||
($line =~ /Women Internet Entrepreneurs/) ||
($line =~ /Goodsol Newsletter/) ||
($line =~ /Contact Fastclick/) ||
($line =~ /Scottsdale Community College/) ||
($line =~ /I will be out of the office/) ||
($line =~ /new at BabyU/) ||
($line =~ /Here is a current Weather Advisory/) ||
($line =~ /MedPulse is a weekly index of key news/) ||
($line =~ /Oxford Journals: Annals of Oncology/) ||
($line =~ /Mail completata con errori permanenti per alcuni/) ||
($line =~ /CERT Advisory/) ||
($line =~ /In an effort to eliminate junk email, I am using Mailwiper/) ||
($line =~ /This e-mail is no longer in use/) ||
($line =~ /inactive email address/) ||
($line =~ /MSNBC News Headlines/) ||
($line =~ /nar.oupjournals.org/) ||
($line =~ /WWW.PSYPHY.ORG/) ||
($line =~ /Your mail to the following recipients could not be delivered/) ||
($line =~ /AWAKENED WOMAN/) ||
($line =~ /Rim Digest eZine/) ||
($line =~ /Autoreply/) ||
($line =~ /POSTGAME ALERT /) ||
($line =~ /The Chef at World Wide Recipes/) ||
($line =~ /Word of the Day/) ||
($line =~ /Weight Room Newsletter/) ||
($line =~ /publisher\@logonnewzine.com/) ||
($line =~ /National Junior Master Gardener Program/) ||
        ($line =~ /I will be on vacation/) ||
        ($line =~ /DIET \& NUTRITION NEWSLETTER/) ||
        ($line =~ /TOPICA - Start your own email discussion group/) ||
        ($line =~ /A copy of the original message/) ||
        ($line =~ /A friend recommends a page from The Onion/) ||
        ($line =~ /A message that you sent could not be delivered/) ||
        ($line =~ /Apartments.com Monthly Property Activity Report/) ||
        ($line =~ /CrossDaily.com The Christian Web/) ||
        ($line =~ /eBay Item Not Won, Similar Items Found/) ||
        ($line =~ /evite.com/) ||
        ($line =~ /Google Email Verification/) ||
        ($line =~ /ITEM NOT WON - SIMILAR ITEMS FOUND/) ||
        ($line =~ /Microsoft for Partners Newsletter/) ||
        ($line =~ /MORE ITEMS FROM THIS SELLER/) ||
        ($line =~ /Notification of an Instant Purchase Payment Received/) ||
        ($line =~ /Plaxo Validation/) ||
        ($line =~ /SIMILAR ITEMS FOUND ON EBAY/) ||
        ($line =~ /Sorry. Your message could not be delivered to/) ||
        ($line =~ /Thank you for using the Principal Residential Mortgage/) ||
        ($line =~ /Thank you for your Auto Bill payment to Verizon Wireless/) ||
        ($line =~ /Thank you for your purchase/) ||
        ($line =~ /Thanks for joining Woody/) ||
        ($line =~ /To release your pending message\(s\) for delivery, please reply to thisThis message is to inform you
that the payment on your Mastercard/) ||
        ($line =~ /Windows Platform News/) ||
        ($line =~ /WINDOWS Watch/) ||
        ($line =~ /Your Mortgage Statement from Principal Residential Mortgage/) ||
        ($line =~ /A message that you sent could not be delivered to one or more/) ||
        ($line =~ /Amazon Order Shipped/) ||
        ($line =~ /Business Online Statement Notification/) ||
        ($line =~ /Books24x7 New Books Notification/) ||
        ($line =~ /City Of Hope National Events Customer Receipt/) ||
        ($line =~ /Fare Watcher Alert/) ||
        ($line =~ /FILM-PHILOSOPHY\@JISCMAIL.AC.UK/) ||
        ($line =~ /Flexnotes/) ||
        ($line =~ /FoolWatch Weekly Digest/) ||
        ($line =~ /I am unavailable to read your message at this time/) ||
        ($line =~ /LinuxQuestions.org /) ||
        ($line =~ /Mail Delivery Problem/) ||
        ($line =~ /mcafeesupport\@mcafee.com/) ||
        ($line =~ /Medscape/) ||
        ($line =~ /Memorymoog-Minimoog-Moog/) ||
        ($line =~ /Microsoft Alumni Network/) ||
        ($line =~ /Please have your ISP\/ASP contact AOL to resolve/) ||
        ($line =~ /Quote.com Portfolio Report/) ||
        ($line =~ /Strategic Forecasting Alert/) ||
        ($line =~ /Tallahassee List/) ||
        ($line =~ /Thank you for using PayPal/) ||
        ($line =~ /This email has been returned as your email address is not recognised by /) ||
        ($line =~ /This mailbox protected from junk email by Matador/) ||
        ($line =~ /turtle-l-digest/) ||
        ($line =~ /Virus found in received message/) ||
        ($line =~ /Yahoo\! Groups/) ||
        ($line =~ /Your Amazon Marketplace order is on its way/) ||
        ($line =~ /Your buy.com order is on its way/) ||
        ($line =~ /Your new United E-Fares/) ||
        ($line =~ /Your Ofoto password is/) ||
        ($line =~ /Astor Piazzolla Mailing List/) ||
        ($line =~ /yahoogroups.com/) ||
        ($line =~ /Mid-Week Market Report/) ||
        ($line =~ /Chicago Sun-Times\: Ebert e-mail edition/) ||
        ($line =~ /LISTS.MSNBC.COM /) ||
        ($line =~ /eudora-emsapi Monthly Help File/) ||
        ($line =~ /Greater Seattle Chamber eNewsletter/) ||
        ($line =~ /Human Rights Action Center Alert List /) ||
        ($line =~ /IBM developerWorks/) ||
        ($line =~ /Java Industry Newsletter/) ||
        ($line =~ /LATimes updaTE/) ||
        ($line =~ /LONGEVITY NEWS/) ||
        ($line =~ /list.themayreport.com /) ||
        ($line =~ /MKTGSTRATEGYSIG/) ||
        ($line =~ /Morbidity and Mortality Weekly Report/) ||
        ($line =~ /Northern California Oracle Users Group/) ||
        ($line =~ /Schwab eStatements/) ||
        ($line =~ /Sesame Family Newsletter/) ||
        ($line =~ /Thank you for using the eFax service/) ||
        ($line =~ /The Cameron Column/) ||
        ($line =~ /This spam was encoded base64.  Normal text is below./) ||
        ($line =~ /Web Services Report from InfoWorld/) ||
        ($line =~ /Yahoo\! Maps/) ||
        ($line =~ /you asked to receive MovieMail from Epinions.com/) ||
        ($line =~ /Your Monthly Account eStatement is ready for viewing/) ||
        ($line =~ /Your Sprint Bill is Ready/) ||
        ($line =~ /Your Webshots Photos/) ||
        ($line =~ /Dear united.com Customer/) ||
        ($line =~ /Thank you for your interest in Travelocity.com/) ||
        ($line =~ /You are now successfully registered at the Los Angeles Times Web site/) ||
        ($line =~ /contact the Mileage Plus Service Center /) ||
        ($line =~ /ESPN.com Fantasy Games/) ||
        ($line =~ /Your account was successfully created/) ||
        ($line =~ /Guru Fantasy Reports, Inc/) ||
        ($line =~ /DAILY GRIST/) ||
        ($line =~ /Art Deadlines List/)
      )
      { # we found a probable non-spam
        $count++;
        $fname =~ s/.txt//;

        my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
$queryStr .= "\"$fname\"";
```

```
  my $sth = $out_dbh->prepare($queryStr);
  $ret = $sth->execute();
  my ($msg_id) = $sth->fetchrow_array();

  if ($msg_id gt "0") { #it's a valid msg id
   print "message file name is $fname msg id is $msg_id\n";
   print "removed by nonspam1 matching line is $line\n";

   # remove message
   $deleteStr = qq{delete FROM Message WHERE \Message_id =};
   $deleteStr .= "\"$msg_id\"";

   my $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();

   $removed_messages = $removed_messages + $ret;
   $sth->finish();

   # remove fingers
   $deleteStr = qq{delete FROM Finger WHERE Finger_Message_id =};
   $deleteStr .= "\"$msg_id\"";

   $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $removed_fingers = $removed_fingers + $ret;
   $sth->finish();

   # remove finger sets
   $deleteStr = qq{delete FROM Finger_set WHERE Finger_set_Message_id =};
   $deleteStr .= "\"$msg_id\"";
   $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $removed_fingersets = $removed_fingersets + $ret;
   $sth->finish();
  } # end if
  goto DONE;
       }
    } # end while
    DONE:
    close IN;
} # end foreach
```

```
#!/usr/local/bin/perl
#!/usr/local/bin/perl -w
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002, 2003
#                       All Rights Reserved
#
#
# Takes a directory of files (with trailing slash) and registers them.
# moves files to CC_nominations or raw_dupes dirs
# created 1-21-03 by Liz Derr
# modified 11-5-03 by LD added call to initialize
# modified 2-15-04 by LD - changed to use PipelineSettings, removed debug prints
##############PIPELINE################
use MIME::Parser;
use MiaVia::PipelineSettings;
use MiaVia::PipelineQueueMimeSpam;
#my $indir = @ARGV;
my $indir = shift (@ARGV);
my $ctype;
my $folder;
MiaVia::PipelineSettings->initialize(@ARGV); # read in the config file, command line args
my $AccessioConfig= MiaVia::PipelineSettings::getConfig();

my $tmpDir = $AccessioConfig->mime_temp();
my $message;
# set up MIME parser
my $parser = MIME::Parser->new();
$parser->output_dir($tmpDir);
$parser->output_to_core(0);
$parser->tmp_to_core(0);
$parser->use_inner_files(0);
opendir INDIR , $indir;
@allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir."/";
      }
my $cur_time = time();
foreach $fname (@allfiles) {

    eval { $message = $parser->parse_open("$indir$fname");};

   if ($@) {
      warn("BatchRegisterSingleFiles file $indir$fname - MIME::Parser error: $!\n");
   } else { #parsed OK
    my $contype = $message->get('Content-Type');
    if (defined($contype)) {

     if($contype =~ /multipart/) { # process multi-part messages
```

```
            my $sub_parts = MiaVia::PipelineQueueMimeSpam::process_multipart_message ($message,1,0); #level=1,
Piece=0
     } else { #else if not multi-part just process as text

        MiaVia::PipelineQueueMimeSpam::process_message($message,"text");
     }
    } else { #else if type not defined just process as text

       MiaVia::PipelineQueueMimeSpam::process_message($message,"text");
    }
    $parser->filer->purge;
    unlink "$indir$fname"; # added 1-26-03 Go Bucs!
 } # end else
 $cur_time=time();


  } # end for each file
```

```
#!/usr/local/bin/perl
use strict;
use MiaVia::Settings;
use MiaVia::DbUtil;
my $count = 0;
my  $removed_fingers =0;
my  $removed_messages =0;
my  $removed_fingersets =0;
my  $ret =0;
my $deleteStr;
my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
my ($indir) = @ARGV;
#print "indir is $indir\n";
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
#print "indir is $indir\n";
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    #print "opening $indir$fname\n";
    while (my $line = <IN>) {
        if (($line =~ /talk-aboutdogs-rescue\@topica.com/) ||
           ($line =~ /Your Ad on www.collectorcartraderonline.com/) ||
           ($line =~ /webmaster\@ohlone.edu/) ||
           ($line =~ /This is a message from the MailScanner E-Mail Virus Protection Service/) ||
           ($line =~ /wsmith\@wordsmith.org/) ||
           ($line =~ /SDForum Events/) ||
           ($line =~ /A.Word.A.Day/) ||
           ($line =~ /The Daily Reckoning/) ||
           ($line =~ /MWave\@subscribermails.com/) ||
           ($line =~ /blackvault\@topica.com/) ||
           ($line =~ /Sorry, no mailbox here by that name/) ||
           ($line =~ /CHABAD.ORG NEWSLETTER/) ||
           ($line =~ /promed-digest-Owner\@promed.isid.harvard.edu (ProMED Digest)/) ||
           ($line =~ /mailing1\@mail.morningstar.net/) ||
           ($line =~ /oup\@oup.co.uk/) ||
           ($line =~ /listowner\@webber.oup.co.uk /) ||
           ($line =~ /NTBUGTRAQ\@LISTSERV.NTBUGTRAQ.COM/) ||
           ($line =~ /alerter\@my-cast.com/) ||
           ($line =~ /nsb\@ysa.org/) ||
           ($line =~ /ScanMail for Microsoft Exchange has taken action on the message/) ||
           ($line =~ /TammyL\@second-to-none.com/) ||
           ($line =~ /From: Microsoft .NET Passport/) ||
           ($line =~ /content violation/) ||
           ($line =~ /entymology/) ||
           ($line =~ /Verisign/) ||
           ($line =~ /Emerging Infectious Diseases Journal Now Online/) ||
($line =~ /California 4-H Youth Development Office/) ||
($line =~ /the cost of shipping to California/) ||
($line =~ /Project Censored/) ||
($line =~ /chembio-terror\@cnslists.miis.edu/) ||
($line =~ /within five days, some of your messages will be automatically deleted/) ||
($line =~ /www.ancestry.com/) ||
($line =~ /WinProxy and eShield Virus Alert/) ||
($line =~ /Comments and Questions (website form)/) ||
($line =~ /joke-of-the-day/) ||
($line =~ /On-Air and Online at AMC/) ||
($line =~ /Apple enews/) ||
($line =~ /Microsoft Business Solution/) ||
($line =~ /Application Spotlight/) ||
($line =~ /Microsoft Windows eHome Division/) ||
($line =~ /Microsoft FYI Strategy magazine/) ||
($line =~ /Newsletters.Microsoft.com/) ||
($line =~ /Newsletter\@HealthandWellnessClub.com/) ||
($line =~ /newsletters\@business2.com/) ||
($line =~ /The PCWorld.com newsletters may contain/) ||
($line =~ /Sondernewsletter /) ||
($line =~ /The FTPplanet.com newsletter is a service/) ||
($line =~ /economist-newsletters-admin\@news.economist.com/) ||
($line =~ /www.earthchangestv.com/) ||
($line =~ /Women Internet Entrepreneurs/) ||
($line =~ /Goodsol Newsletter/) ||
($line =~ /Contact Fastclick/) ||
($line =~ /Scottsdale Community College/) ||
($line =~ /I will be out of the office/) ||
($line =~ /new at BabyU/) ||
($line =~ /Here is a current Weather Advisory/) ||
($line =~ /MedPulse is a weekly index of key news/) ||
($line =~ /Oxford Journals: Annals of Oncology/) ||
($line =~ /Mail completata con errori permanenti per alcuni/) ||
($line =~ /CERT Advisory/) ||
($line =~ /In an effort to eliminate junk email, I am using Mailwiper/) ||
($line =~ /This e-mail is no longer in use/) ||
($line =~ /inactive email address/) ||
($line =~ /MSNBC News Headlines/) ||
($line =~ /nar.oupjournals.org/) ||
($line =~ /WWW.PSYPHY.ORG/) ||
($line =~ /Your mail to the following recipients could not be delivered/) ||
($line =~ /AWAKENED WOMAN/) ||
($line =~ /Rim Digest eZine/) ||
($line =~ /Autoreply/) ||
($line =~ /POSTGAME ALERT /) ||
($line =~ /The Chef at World Wide Recipes/) ||
($line =~ /Word of the Day/) ||
($line =~ /Weight Room Newsletter/) ||
($line =~ /publisher\@logonnewzine.com/) ||
($line =~ /National Junior Master Gardener Program/) ||
($line =~ /I will be on vacation/) ||
($line =~ /DIET \& NUTRITION NEWSLETTER/) ||
($line =~ /TOPICA - Start your own email discussion group/) ||
($line =~ /A copy of the original message/) ||
($line =~ /A friend recommends a page from The Onion/) ||
($line =~ /A message that you sent could not be delivered/) ||
($line =~ /Apartments.com Monthly Property Activity Report/) ||
($line =~ /eBay Item Not Won, Similar Items Found/) ||
($line =~ /evite.com/) ||
($line =~ /Google Email Verification/) ||
($line =~ /ITEM NOT WON - SIMILAR ITEMS FOUND/) ||
($line =~ /Microsoft for Partners Newsletter/) ||
($line =~ /MORE ITEMS FROM THIS SELLER/) ||
($line =~ /Notification of an Instant Purchase Payment Received/) ||
($line =~ /Plaxo Validation/) ||
($line =~ /SIMILAR ITEMS FOUND ON EBAY/) ||
($line =~ /Sorry. Your message could not be delivered to/) ||
($line =~ /Thank you for using the Principal Residential Mortgage/) ||
($line =~ /Thank you for your Auto Bill payment to Verizon Wireless/) ||
($line =~ /Thank you for your purchase/) ||
($line =~ /Thanks for joining Woody/) ||
($line =~ /To release your pending message/) ||
($line =~ /Windows Platform News/) ||
($line =~ /WINDOWS Watch/) ||
($line =~ /Your Mortgage Statement from Principal Residential Mortgage/) ||
($line =~ /Business Online Statement Notification/) ||
($line =~ /Books24x7 New Books Notification/) ||
($line =~ /City Of Hope National Events Customer Receipt/) ||
($line =~ /Fare Watcher Alert/) ||
($line =~ /FILM-PHILOSOPHY\@JISCMAIL.AC.UK/) ||
($line =~ /Flexnotes/) ||
($line =~ /FoolWatch Weekly Digest/) ||
($line =~ /I am unavailable to read your message at this time/) ||
($line =~ /LinuxQuestions.org /) ||
($line =~ /Mail Delivery Problem/) ||
($line =~ /mcafeesupport\@mcafee.com/) ||
($line =~ /Medscape/) ||
($line =~ /Memorymoog-Minimoog-Moog/) ||
($line =~ /Microsoft Alumni Network/) ||
($line =~ /Please have your ISP\/ASP contact AOL to resolve/) ||
($line =~ /Quote.com Portfolio Report/) ||
($line =~ /Strategic Forecasting Alert/) ||
($line =~ /Tallahassee List/) ||
($line =~ /Thank you for using PayPal/) ||
($line =~ /This email has been returned as your email address is not recognised by /) ||
($line =~ /This mailbox protected from junk email by Matador/) ||
($line =~ /turtle-l-digest/) ||
($line =~ /Virus found in received message/) ||
($line =~ /groups.google.com/) ||
($line =~ /tw.bbs.music.classical/) ||
($line =~ /www.rimdigest.com/) ||
($line =~ /WirelessWorld\@bdcimail.com/) ||
($line =~ /NStorm Game News/) ||
($line =~ /PCWorld.com/) ||
($line =~ /InformationWeek/) ||
($line =~ /Topica Digest/) ||
($line =~ /California Businesses For Sale/) ||
($line =~ /WinXPnews/) ||
($line =~ /Equifax Personal/) ||
($line =~ /Recipes4Living/) ||
($line =~ /Declude Anti-Virus software/) ||
($line =~ /Dear Valued Go Daddy Customer/) ||
($line =~ /dailyhoroscope\@astrology.com/) ||
($line =~ /Larta.org/) ||
($line =~ /A Virus was found in an Email message/) ||
($line =~ /CreditMatters/) ||
($line =~ /SmallCap MarketWatch/) ||
($line =~ /News\@eDiets/) ||
($line =~ /BabyUniversity/) ||
($line =~ /Dollar Stretcher Tips/) ||
($line =~ /BabyCenter Bulletin/) ||
($line =~ /CrossDaily.com/) ||
($line =~ /Columbia House DVD Club/) ||
($line =~ /support\@rent.com/) ||
($line =~ /Broker Agent News/) ||
($line =~ /promo.gateway.com/) ||
($line =~ /Engaging News/) ||
($line =~ /Inman News Headlines/) ||
($line =~ /SCIFI.COM WEEKLY NEWS/) ||
($line =~ /Travelocity.com/) ||
($line =~ /SchwabLearning.org/) ||
($line =~ /BioSpace Breaking News/) ||
($line =~ /EveryMac.com/) ||
($line =~ /MP3.com Music Newsletter/) ||
($line =~ /fromthehomefront.com/) ||
($line =~ /FAILURE TO RENEW YOUR DOMAIN BEFORE/) ||
($line =~ /channel4000/) ||
($line =~ /fit:perks(TM) Offer/) ||
($line =~ /www.homeownertips.com/) ||
($line =~ /MAIL.DOE.STATE.FL.US/) ||
($line =~ /Natural Family Home Newsletter/) ||
($line =~ /Disney Insider/) ||
($line =~ /WebReference/) ||
($line =~ /Tipz Time/) ||
($line =~ /Nature Science Update/) ||
($line =~ /backtothebible.org/) ||
($line =~ /newsletter\@carprices.com/) ||
($line =~ /egroups.rediff.com/) ||
($line =~ /forresterresearch.01o.net/) ||
($line =~ /www.zoompanel.com/) ||
($line =~ /thefamilycorner.com/) ||
($line =~ /editor\@kmworld.com/) ||
($line =~ /Broker Agent News/) ||
($line =~ /BabyShopsOnline/) ||
($line =~ /Handspring e-newsletter/) ||
($line =~ /HFNetChkLT/) ||
($line =~ /I do not already have your email address in my whitelist/) ||
($line =~ /e-showtimes/) ||
($line =~ /Dear Amazon.com Customer/) ||
($line =~ /www.emazing.com/) ||
($line =~ /Planned Parenthood Federation/) ||
($line =~ /see.sun.com/) ||
($line =~ /Joke-Of-The-Day/) ||
($line =~ /Reminder from the Calendar/) ||
($line =~ /news.economist.com/) ||
($line =~ /Your Webshots Photos/) ||
($line =~ /Amazon Order Shipped/) ||
($line =~ /Your Amazon Marketplace order is on its way/) ||
($line =~ /Your buy.com order is on its way/) ||
($line =~ /Your new United E-Fares/) ||
($line =~ /Your Ofoto password is/) ||
($line =~ /Astor Piazzolla Mailing List/) ||
($line =~ /yahoogroups.com/) ||
($line =~ /Mid-Week Market Report/) ||
($line =~ /Chicago Sun-Times\: Ebert e-mail edition/) ||
($line =~ /LISTS.MSNBC.COM /) ||
($line =~ /eudora-emsapi Monthly Help File/) ||
($line =~ /Greater Seattle Chamber eNewsletter/) ||
($line =~ /Human Rights Action Center Alert List /) ||
($line =~ /IBM developerWorks/) ||
($line =~ /Java Industry Newsletter/) ||
($line =~ /LATimes updaTE/) ||
($line =~ /LONGEVITY NEWS/) ||
($line =~ /list.themayreport.com /) ||
($line =~ /MKTGSTRATEGYSIG/) ||
($line =~ /Morbidity and Mortality Weekly Report/) ||
($line =~ /Northern California Oracle Users Group/) ||
($line =~ /Schwab eStatements/) ||
($line =~ /Sesame Family Newsletter/) ||
($line =~ /Thank you for using the eFax service/) ||
($line =~ /The Cameron Column/) ||
($line =~ /This spam was encoded base64.  Normal text is below./) ||
($line =~ /Web Services Report from InfoWorld/) ||
($line =~ /Yahoo\! Maps/) ||
($line =~ /you asked to receive MovieMail from Epinions.com/) ||
($line =~ /Your Monthly Account eStatement is ready for viewing/) ||
($line =~ /Your Sprint Bill is Ready/) ||
($line =~ /Dear united.com Customer/) ||
($line =~ /Thank you for your interest in Travelocity.com/) ||
($line =~ /You are now successfully registered at the Los Angeles Times Web site/) ||
         ($line =~ /contact the Mileage Plus Service Center /) ||
         ($line =~ /ESPN.com Fantasy Games/) ||
         ($line =~ /Your account was successfully created/) ||
         ($line =~ /Guru Fantasy Reports, Inc/) ||
         ($line =~ /DAILY GRIST/) ||
         ($line =~ /Art Deadlines List/))
       { # we found a probable non-spam
         $count++;
         $fname =~ s/.txt//;

         my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
 $queryStr .= "\"$fname\"";
 my $sth = $out_dbh->prepare($queryStr);
 $ret = $sth->execute();
 my ($msg_id) = $sth->fetchrow_array();

 if ($msg_id gt "0") { #it's a valid msg id
  print "message file name is $fname msg id is $msg_id\n";
  print "removed by nonspam2 matching line is $line\n";

  # remove message
  $deleteStr = qq{delete FROM Message WHERE \Message_id =};
  $deleteStr .= "\"$msg_id\"";

  my $sth = $out_dbh->prepare($deleteStr);
  $ret = $sth->execute();

  $removed_messages = $removed_messages + $ret;
  $sth->finish();

  # remove fingers
  $deleteStr = qq{delete FROM Finger WHERE Finger_Message_id =};
  $deleteStr .= "\"$msg_id\"";

  $sth = $out_dbh->prepare($deleteStr);
  $ret = $sth->execute();
  $removed_fingers = $removed_fingers + $ret;
  $sth->finish();

  # remove finger sets
  $deleteStr = qq{delete FROM Finger_set WHERE Finger_set_Message_id =};
  $deleteStr .= "\"$msg_id\"";
  $sth = $out_dbh->prepare($deleteStr);
  $ret = $sth->execute();
  $removed_fingersets = $removed_fingersets + $ret;
  $sth->finish();
 } # end if
 goto DONE;
       }
    } # end while
    DONE:
    close IN;
} # end foreach
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# created 12-30-3 by LD
# 2-15-04 clear of Settings references
# $Id: SubLink.pm,v 1.9 2004/02/16 03:24:21 lizd Exp $
package MiaVia::SubLink;
use strict;
## CLASS METHODS
##
# Create a new SubLink entity, given a list of sub links
#
sub new {
  my ($class,             # SubLink
     $subElemREF) = @_;  # list of sub links
  my @sublinks = ();
  my $self = {  sublink_list => \@sublinks
            };
  foreach my $subElem (@{$subElemREF}) { # each sublink in the list
     push @sublinks, $subElem;
  }
  bless $self, $class;
}
## INSTANCE METHODS
##
sub sublink_list {
  my $self = shift;
  return $self->{sublink_list};

}
1; # end of Sublink.pm
```

```
INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("com",NOW());

INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("htm",NOW());

 INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("cgi",NOW());

 INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("html",NOW());

 INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("www",NOW());

 INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("net",NOW());

 INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("org",NOW());

  INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("jpg",NOW());

  INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("gif",NOW());

   INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("asp",NOW());

    INSERT INTO CommonSubLink (CommonSubLink_sub_string,  CommonSubLink_inserted_on)
 VALUES("biz",NOW());
```

```
#!/usr/local/bin/perl
use strict;
use MiaVia::Settings;
use MiaVia::DbUtil;
my $count = 0;
my  $removed_fingers =0;
my  $removed_messages =0;
my  $removed_fingersets =0;
my  $ret =0;
my $deleteStr;
my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
my ($indir) = @ARGV;
#print "indir is $indir\n";
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
#print "indir is $indir\n";
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    #print "opening $indir$fname\n";
    while (my $line = <IN>) {
        if (($line =~ /Amazon Marketplace/) ||
($line =~ /Ancestry Weekly Digest/) ||
($line =~ /apple\@listas.apple.com.br/) ||
($line =~ /Bay Area Bad Weather Hiking Group/) ||
($line =~ /Bedava\.org/) ||
($line =~ /Dear united.com Customer/) ||
($line =~ /did not reach the following recipient/) ||
#($line =~ /vite.com/) ||
($line =~ /feedback\@segway.com/) ||
($line =~ /Freaky Freddies Free Stuff/) ||
# ($line =~ /half.com/) ||
($line =~ /I have a retail mystery shop available in your area./) ||
($line =~ /IBM WebSphere and e-business on Demand Competitive Technical Briefing/) ||
($line =~ /I'm afraid I wasn't able to deliver your message to the following addresses./) ||
($line =~ /In an effort to eliminate junk email, I am using Mailwiper/) ||
($line =~ /is to inform you that the payment on your Mastercard/) ||
($line =~ /LinkAlarm checked the links on the site/) ||
($line =~ /Mailbox temporarily disabled/) ||
($line =~ /March Hare: Issue/) ||
($line =~ /NorthBayMunch\@bigfoot.com/) ||
($line =~ /Out of Office AutoReply/) ||
($line =~ /pcwwatch\@listproc.pcworld.com/) ||
($line =~ /Please reactivate your Yahoo! Groups account/) ||
($line =~ /Prayer Request/) ||
($line =~ /PSYPHY/) ||
($line =~ /Second To None ENTHUSIASTICALLY supports certification by the MSPA./) ||
($line =~ /Sw-discuss/) ||
($line =~ /SwiftPay UserID/) ||
($line =~ /Thank you for using Pictopia./) ||
($line =~ /The following postings have been made on FBO/) ||
($line =~ /The May Report/) ||
($line =~ /the_edge\@macromedia.com/) ||
($line =~ /This is the qmail/) ||
($line =~ /webber.oup.co.uk/) ||
($line =~ /WFANPROMOTIONS\@WFAN.COM/) ||
($line =~ /Wired News Daily/) ||
($line =~ /www.DecorDecoratingAndDesign.com/) ||
($line =~ /www.logonnewzine.com/) ||
($line =~ /www.sdforum.org/) ||
($line =~ /subscribe.yahoo.com/) ||
($line =~ /Yahoo\! Groups/) ||
($line =~ /You are subscribed to OpenSource IT Industry Update as/) ||
           ($line =~ /YOUR DAILY INSPIRATIONS FROM QUICKINSPIRATIONS.COM/))
        { # we found a probable non-spam
          $count++;
          $fname =~ s/.txt//;

          my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
  $queryStr .= "\"$fname\"";
  my $sth = $out_dbh->prepare($queryStr);
  $ret = $sth->execute();
  my ($msg_id) = $sth->fetchrow_array();

  if ($msg_id gt "0") { #it's a valid msg id
   print "message file name is $fname msg id is $msg_id\n";
   print "removed by nonspam3 matching line is $line\n";
   # remove message
   $deleteStr = qq{delete FROM Message WHERE \Message_id =};
   $deleteStr .= "\"$msg_id\"";

   my $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();

   $removed_messages = $removed_messages + $ret;
   $sth->finish();

   # remove fingers
   $deleteStr = qq{delete FROM Finger WHERE Finger_Message_id =};
   $deleteStr .= "\"$msg_id\"";

   $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $removed_fingers = $removed_fingers + $ret;
   $sth->finish();

   # remove finger sets
```

```
    $deleteStr = qq{delete FROM Finger_set WHERE Finger_set_Message_id =};
    $deleteStr .= "\"$msg_id\"";
    $sth = $out_dbh->prepare($deleteStr);
    $ret = $sth->execute();
    $removed_fingersets = $removed_fingersets + $ret;
    $sth->finish();
   } # end if
   goto DONE;
         }
      } # end while
      DONE:
      close IN;
} # end foreach
```

```
#!/usr/local/bin/perl
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2003
#                       All Rights Reserved
#
# This program parses the log files created by accessiod
# created 7-26-03 by Liz Derr
use strict;
use MiaVia::PipelineUtils;
use MiaVia::DbUtil;
my ($fname) = @ARGV;
my %queue_msgs;
my ($stuff, $date, $dtime, $reg, $comment, $remainder );
my ($one, $two, $of, $msgID, $three, $score, $file, $four, $seencount);
open IN, $fname or die("can't open input file $fname: $!");
while (my $line = <IN>) {
 ($stuff, $reg, $msgID, $remainder )
 #($date, $dtime, $reg, $comment, $remainder )
  = split(/ /, $line);
  #print "comment is $comment\n";
  #if (length($comment)>0){
 # ($one, $two, $of, $msgID, $three, $score, $file, $four, $seencount)
 # = split(/ /, $comment);
 # print "one is $one two is $two, msgID is $msgID seen count is $seencount\n\n";
 #} else {
 # print "date is $date, time is $dtime, reg is $reg, remainder is $remainder\n";
 #}

#QDB DUPE of 397389 score: 82.5708061002179
/usr/local/accessio/data/queue_dupes/66be706b15a64c623245459dd3404eab.txt seenCount: 5
# HANDPRINT DUPLICATE of 407941 scored 49.552071668533
/usr/local/accessio/data/raw_dupes/6da11e947a679fdcc3e37f73fda33909.txt
  # print "reg is $reg and msg id is $msgID";
  #if ($one =~ /QBD/) {
  if ($reg =~ /REGISTERQ/) {
       if ( $msgID ne "") {
              $queue_msgs{$msgID}++;
             }
        }
}# end while
close IN;
my $queue_dbh = MiaVia::PipelineUtils::openQueueDb('miavia_wrtr');
my $file_name;
my $mydoom_count=0;
my $nonspam_count = 0;
my $problem_count= 0;
foreach my $key (sort hashValueDescendingNum (keys(%queue_msgs))) {
   $file_name = "";
   my $num = $queue_msgs{$key};
   if (          ($key eq "123514")
     || ($key eq "172142")
     || ($key eq "133682")
     || ($key eq "128965")
     || ($key eq "128268")) {

     $mydoom_count = $mydoom_count + $num;
   } elsif (      ($key eq "87347")
     || ($key eq "6384")
     || ($key eq "1688")
     || ($key eq "208658")
     || ($key eq "179492")
     || ($key eq "293340")
     || ($key eq "285988")
     || ($key eq "220017")
     || ($key eq "204661")
     || ($key eq "281146")
     || ($key eq "270142")
     || ($key eq "226744")
     || ($key eq "221125")
     || ($key eq "217082")
     || ($key eq "8540")
     || ($key eq "8455")
     || ($key eq "77844")
     || ($key eq "296843")
     || ($key eq "8699")

     ) {
     $nonspam_count = $nonspam_count + $num;
     } elsif (  ($key eq "88319")
       || ($key eq "275310")
     ) {
     $problem_count = $problem_count + $num;
   } elsif ($num > 6) { # if we've seen a few of these
    $file_name = MiaVia::DbUtil::getMessage_filename($key,$queue_dbh);
        if (length($file_name) > 2) { # we got one that hasn't been inserted into spamdb
     print "queued_msg: $key num messages: $num filename: $file_name\n";
    }
   }
}
print "\n$mydoom_count messages related to MyDoom virus seen so far today\n";
print "$nonspam_count known non spam messages seen so far today\n";
print "$problem_count known problem messages seen so far today\n";
sub hashValueDescendingNum {
 $queue_msgs{$b} <=> $queue_msgs{$a};
}
```

```
#!/usr/local/bin/perl
use strict;
use CGI;
my $q = new CGI;
my $phrase = "";
print $q->header("text/html"),
    $q->start_html(-title => "Scrub Nominations List", -bgcolor =>"#ffffff");
print $q->start_form( -method => "get", -action => "Ming_scrub.cgi");
print "<p><b>How should matching Messages be marked?</b></P><p>";
print $q->radio_group( -name => "Marked",
                -values => ["S", "N", "P"],
                -default => "S",
                -labels => { S => "Spam", N => "Not Spam", P => "Problems"}
               );
print "</p>";

print $q->p($q->submit( -value => "submit"));
print "<p> This will mark all files in the nominations list that contain this phrase.</p><p>";
print "<p> Note: the phrase must be at least 8 characters long.</p><p>";
print $q->textfield(-name => 'phrase', -label =>'Phrase to scrub: ', -value => $phrase, -size =>80);
print "</p>";
print $q->end_form;
print $q->end_html;
```

```
#!/usr/local/bin/perl
use strict;
use MiaVia::Settings;
use MiaVia::DbUtil;
my $count = 0;
my  $removed_fingers =0;
my  $removed_messages =0;
my  $removed_fingersets =0;
my  $ret =0;
my $deleteStr;
my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
my ($indir) = @ARGV;
#print "indir is $indir\n";
opendir INDIR , $indir;
my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
closedir INDIR;
if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
}
#print "indir is $indir\n";
foreach my $fname (@allfiles) {
    open IN, "$indir$fname";
    #print "opening $indir$fname\n";
    while (my $line = <IN>) {
        if (($line =~ /hotjobs.yahoo.com/) ||
($line =~ /Dilbert/) ||
($line =~ /ecardview.hallmark.com/) ||
($line =~ /Reminder from the/) ||
($line =~ /startribune.com/) ||
($line =~ /errors occurred during message delivery/) ||
($line =~ /sdforum.org/) ||
($line =~ /InfoWorld.com/) ||
($line =~ /Consular Information Sheet/) ||
($line =~ /INFOWORLD.COM/) ||
($line =~ /WWW.SUPERMEDIASTORE.COM/) ||
($line =~ /calendar.yahoo.com/) ||
($line =~ /India News Network/) ||
($line =~ /This Week on NW Fusion/) ||
($line =~ /www.naesp.org/) ||
($line =~ /Sportstextiles Weekly Newsletter/) ||
($line =~ /National Service Briefing/) ||
($line =~ /Mommysavers Newsletter/) ||
($line =~ /adviceby.com/) ||
($line =~ /Foghead/) ||
($line =~ /EndMail/) ||
($line =~ /PartnerWorld/) ||
($line =~ /bankofamerica1/) ||
($line =~ /alumniconnections/) ||
($line =~ /myfloridahouse/) ||
($line =~ /24hourfitness/) ||
($line =~ /officefurniture/) ||
($line =~ /Announcement\@ohlone.cc.ca.us/) ||
($line =~ /newvibe.com/) ||
($line =~ /platinum.yahoo.com/) ||
($line =~ /eAds Plus e-zine/) ||
($line =~ /Clearance Center Weekly Newsletter/) ||
($line =~ /This Week on NW Fusion/) ||
($line =~ /This person has asked to be SUBSCRIBED to the newsletter list/) ||
($line =~ /Your request to join our E-Mail List/) ||
($line =~ /ZDNet newsletters/) ||
($line =~ /completeresults/) ||
($line =~ /executechfinancialadvisors/) ||
($line =~ /Job Seeker News/) ||
($line =~ /American Express Corporate Vacations/) ||
($line =~ /support.bluemountain/) ||
($line =~ /newsletter\@jhblive.com/) ||
($line =~ /ChartJungle News/) ||
($line =~ /Bargains Depot newsletter/) ||
($line =~ /cbs.marketwatch/) ||
($line =~ /This communication is sent by your request for more/) ||
($line =~ /W2Knews/) ||
($line =~ /moneydots.com/) ||
($line =~ /majordomo.law.com/) ||
($line =~ /astrocenter.com/) ||
#($line =~ /groups.yahoo.com/) ||
($line =~ /X-eBay-MailTracker/) ||
($line =~ /\@yahoogroups.com/) ||
($line =~ /yahoo-delivers\@yahoo-inc.com/) ||
($line =~ /verisonwireless.com/) ||
($line =~ /www.buy.com/) ||
($line =~ /www.half.com/) ||
($line =~ /McAfee Dispatch/) ||
($line =~ /\@ebay.com/) ||
($line =~ /\@emailebay.com/) ||
($line =~ /\@reply1.ebay.com/) ||
($line =~ /\@reply2.ebay.com/) ||
($line =~ /\@reply3.ebay.com/) ||
($line =~ /\@ebaydns.com/) ||
($line =~ /\@paypal.com/) ||
($line =~ /\@bidpay.com/) ||
($line =~ /\@evite.com/) ||
($line =~ /VersionTracker/))
        { # we found a probable non-spam
          $count++;
          $fname =~ s/.txt//;

          my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
  $queryStr .= "\"$fname\"";
  my $sth = $out_dbh->prepare($queryStr);
  $ret = $sth->execute();
  my ($msg_id) = $sth->fetchrow_array();

  if ($msg_id gt "0") { #it's a valid msg id
   print "message file name is $fname msg id is $msg_id\n";
   print "removed by nonspam4 matching line is $line\n";
   # remove message
   $deleteStr = qq{delete FROM Message WHERE \Message_id =};
   $deleteStr .= "\"$msg_id\"";

   my $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();

   $removed_messages = $removed_messages + $ret;
   $sth->finish();

   # remove fingers
   $deleteStr = qq{delete FROM Finger WHERE Finger_Message_id =};
   $deleteStr .= "\"$msg_id\"";

   $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $removed_fingers = $removed_fingers + $ret;
   $sth->finish();

   # remove finger sets
   $deleteStr = qq{delete FROM Finger_set WHERE Finger_set_Message_id =};
   $deleteStr .= "\"$msg_id\"";
   $sth = $out_dbh->prepare($deleteStr);
   $ret = $sth->execute();
   $removed_fingersets = $removed_fingersets + $ret;
   $sth->finish();
  } # end if
  goto DONE;
       }
    } # end while
    DONE:
    close IN;
} # end foreach
```

```
#!/usr/local/bin/perl
# created 5-18-04 by Liz Derr
# take the messages in indir (presumed to be inspected false
# positives) and send them to the false positive reporting
# email address for the X-Accessio-Recipient as specified
# in the fpos_report.cnf config file. Log the transaction
# in the fpos log file.
use strict;
use MIME::Parser;
use MiaVia::PipelineSettings;
use MiaVia::PipelineLogEvent;
use AppConfig qw( :argcount :expand );
use AppConfig::State;
# Set up the config settings, read the fpos report config file,
# move files to the work dir, set up the logging
  # get dir params
  my ($indir, $workdir, $outdir) = @ARGV;
  die "no indir defined" unless defined $indir;
  die "no workdir defined" unless defined $workdir;
  die "no outdir defined" unless defined $outdir;
  if (!($indir =~/\/$/)) {
 $indir = $indir.'/';
  }
  if (!($workdir =~/\/$/)) {
 $workdir = $workdir.'/';
  }
  if (!($outdir =~/\/$/)) {
 $outdir = $outdir.'/';
  }
  # config settings
  my $AccessioConfig= MiaVia::PipelineSettings::getConfig();
  my $mv_home = $AccessioConfig->mv_home();

  # Setup the logging
  my $klogger = new PipelineLogEvent(
      TYPE => 'fpos'
    , LOG_DIR => "$mv_home/log/"
   , PATH_TO_CRONOLOG => "$mv_home/sbin/mvlogger"
  );

  # open known_spam and get all the file names from the directory,
  # except the dots
  opendir INDIR , $indir;
  my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
  closedir INDIR;

  #fpos_report config file
  my $filename = "$mv_home/fpos_report.cnf";
  if(! -e $filename) {
 # No user config file found, abort
 warn("no config file found");
        return 0;
 }
 my $state = AppConfig::State->new ( {CREATE => 1});

 my $config = AppConfig->new({
  GLOBAL => {DEFAULT => "<undef>"}}) or return 0;

 $config->define("report", {
    DEFAULT   => "<undef>"
  , ARGCOUNT => ARGCOUNT_HASH
});

 $config->file("$filename");
 my $reportHashREF = $config->report();

 # set up the mime parser
 my $tmpDir = $mv_home.'/data/tmp';
 if (!-d $tmpDir) {
  warn "MIME Processor: MIME temp directory does not exist: $tmpDir\n";
 }
 my $parser = MIME::Parser->new();
 $parser->output_dir($tmpDir);
 $parser->output_to_core(0);
 $parser->tmp_to_core(0);
 $parser->use_inner_files(0);

 # move files to the work dir
 foreach my $fname (@allfiles) {
   rename  "$indir$fname","$workdir$fname"
     or warn ("can't rename file $indir$fname to $workdir$fname: $!");
 } # end for each
 # now, process each file that we moved to the work dir
 foreach my $fname (@allfiles) {
      # parse the file
      my $entity = $parser->parse_open("$workdir$fname")
       or warn("can't open parser for $workdir$fname: $!");
  my $head = $entity->head;
  my $recipient = $head->get('X-Accessio-Recipient',0); # get the first one

  $recipient =~ s/\(m-Id\)//;
  $recipient =~ s/\(r-ap\)//;
  $recipient =~ s/\s//; # remove white space
  $recipient =~ s/\n//; # removenewline
  my $report_addr = undef;
$report_addr = $reportHashREF->{$recipient};
 my $Subject = $head->get("Subject");
 my $subject = $Subject;
 $subject =~ s/\"//g;
 $subject =~ s/\'//g;
```

```
  $subject =~ s/\`//g;
  $subject =~ s/\!//g;

  rename  "$workdir$fname", "$outdir$fname"
       or warn ("can't rename file $workdir$fname to $outdir$fname: $!");
  # if we have a reporting address, mail the file to it
       if ($report_addr) {
       #system ("/usr/bin/mutt", "-s False Positive Report: $subject",  "-a /usr/local/accessio/$outdir$fname",
"$report_addr" );

       #open MAILTO, "|mutt -s \"False Positive Report: $subject\" -a \"/usr/local/accessio/$outdir$fname\" $report_addr";
  #print MAILTO "false positive message attached\n";
  #open MAILTO, "|mailx -s \"False Positive Report: $subject\" $report_addr";
  #print MAILTO $entity->stringify;
  #close MAILTO;

  my $top = MIME::Entity->build(Type    =>"multipart/mixed",
                                From    => "miavia\@miavia.com",
                                To      => "$report_addr",
                                Subject => "False Positive Report: $subject");
              #$top->attach(Data=>"False positive message attached\n");
              #$top->attach(Data=>$entity, Type => "message");
              # $top->attach(Path=>"/usr/local/accessio/$outdir$fname");

              # Build container object for original message
    my $container   = MIME::Entity->build(
     Type => 'message/rfc822',
      Data => [ "" ]
  );
  # Add original message to container entity...
  $container->add_part($entity);

  # Add the container
  $top->add_part($container);

  open MAIL, "| /usr/lib/sendmail -t -oi -oem" or die "open: $!";
          $top->print(\*MAIL);
         close MAIL;
     } else {
       $report_addr = "not_reported";
     }

       #print "recipient is $recipient and addr is $report_addr\n";
       # move the file out of the work dir

       # log the message transaction
       my $MessageId = $head->get("Message-ID"); # this is the email message id
   my $Sender = $head->get("Sender");
   my $Date = $head->get("Date");
   my $To = $head->get("To");
```

```
 my $From = $head->get("From");

 $klogger->NewlogFposEvent(
        STATUS => $recipient # the spam pot address that the fpos came in on
      , SENDER => $Sender
      , FROM => $From
      , TO => $report_addr  # the address the report was sent to
      , SUBJECT => $Subject
      , MAIL_MSG_ID => $MessageId # this is the email message id
      , MAIL_DATE => $Date
      , FILENAME => $fname
     );

   $parser->filer->purge; # very important to purge this!
}
```

```
#!/bin/bash
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# scrubs files prior to inspection
PERL=/usr/local/bin/perl
BINDIR=/usr/local/accessio/bin
DATADIR=/usr/local/accessio/data
STAGING_DIR=/usr/local/accessio/data/CC_nominations
TARGET_DIR=/usr/local/accessio/data/nominations
PERL5LIB=/usr/local/accessio/lib
export PERL5LIB
# remove files that crash Jeffs computer
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/removed_from_queue "<href=http://hotmail.com/>"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/removed_from_queue "Test, yep."
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/removed_from_queue "indows-1252"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/removed_from_queue "Unicode characters"
# remove likely non-spam
$PERL $BINDIR/RemoveProbableNonSpam.pl $STAGING_DIR $DATADIR/probable_nonspam
# remove probable uncatchable repeating spam
$PERL $BINDIR/RemoveUncatchableSpam.pl $STAGING_DIR $DATADIR/probable_duplicate_spam
# remove uncatchable asian/cyrillic spam
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/windows-1251 "charset=windows-1251"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/windows-1251 "charset=Windows-1251"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/koi8-r "koi8-r"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/big5 "big5"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/gb2312 "GB2312"
$PERL $BINDIR/MoveByString.pl $STAGING_DIR $DATADIR/gb2312 "gb2312"
# wait for perl processes to finish
for filename in $STAGING_DIR/*.txt
do
	mv $filename $TARGET_DIR
done
#exit 0
```

```
#!/bin/bash
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#
#
# updated 4-17-04 (Happy 70th, Mom!) to get count from the database instead
DATECMD=/bin/date
PERL=/usr/local/bin/perl
BINDIR=/usr/local/accessio/bin
PERL5LIB=/usr/local/accessio/lib
export PERL5LIB
DAYDATE=`"$DATECMD" "+%Y-%m-%d"`
#LOGFILE=/usr/local/accessio/log/batchInsert${DAYDATE}.log
#INSERTDIR=/usr/local/accessio/data/known_spam
#DAYDATE=`"$DATECMD" "+%m %d %Y"`
#TOTALYES=`grep -c Message $LOGFILE`
#TOTALNEW=`grep -c original $LOGFILE`
#TOTALDUPE=`grep -c "is a duplicate" $LOGFILE`
#HIGHCOUNT=`grep -c "####count" $LOGFILE`
#FILESLEFT=`ls -1 $INSERTDIR | wc -l`
FILESINSERTED=`$PERL $BINDIR/InsertCount.pl $DAYDATE`
#echo "Batch Insert Report for ${DAYDATE}"
#echo "${TOTALNEW} new messages were inserted into db"
#echo "${TOTALDUPE} were duplicates"
#echo "${TOTALYES} total messages marked Yes at inspection"
echo "${FILESINSERTED}" | mail -s "${DAYDATE} Daily Spam Insert report" insert_report@miavia.com
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: MessageTag.pm,v 1.46 2004/02/28 10:24:40 lizd Exp $
#
# changed 8/28/03 by LD - added new format, etc for X-headers
# 2-15-04 clear of all Settings references
# 2-19-04 added category, can calls to Utils to get settings
package MiaVia::MessageTag;
## MessageTag creates the X-Accessio email header tag
use MiaVia::NewScore;
use MiaVia::Message;
use MiaVia::Utils;
use MiaVia::DbUtil;
use MiaVia::SysLogEvent qw(debug_msg);
sub createXAccessioTag {
  my ($message, $score, $dbh) = @_;
  my $tag;
  my $match="none";

  if (!(defined $message)) {
   debug_msg("message object is undefined");
  }
  if (!(defined $score)) {
   debug_msg("score object is undefined");
  }
  my $pass_score = MiaVia::Utils->get_pass_score();
  my $categorize = MiaVia::Utils->get_categorize();
  my $threshold =  MiaVia::Utils->get_x_threshold_tag();
# get the score and matching message number
  my $spamScore = $score->getScore();

  if (defined $spamScore) {
    $match = $score->getMatchId();
   debug_msg("spam score is $spamScore and match is $match");
  } else {
   debug_msg("spam score is undefined");
  }

  if ($spamScore >= $pass_score) {
   $tag = "YES, "
  } else {
   $tag = "NO, "
  }

# obscure the message score
#  $spamScore = $spamScore/10;

  # put in the score
  my $score_string = sprintf("score=%.2f,",$spamScore);
  if (defined($score_string)) {
    $tag = $tag."$score_string";
  } else {
   $tag = $tag."score=000000,"; # flag a message without a defined score
   debug_msg("score string is undefined");
  }

  # put the message ID after the comma in the score
  if (defined($match)) {
    $tag = $tag."$match ";
  } else {
   $tag = $tag."none ";
   debug_msg("match id is undefined");
  }

  # put in the threshold, if configured to
  if ($threshold) {
   $ tag = $tag . "threshold=$pass_score ";
  }

  # put in the category, if configured to
  if (($categorize) && ($spamScore >=$pass_score)) {
   my $category = $score->getCategory();
   my $cat_string = MiaVia::DbUtil::getCatString($dbh, $category);
   if (defined $cat_string) {
    $ tag = $tag . "category=$cat_string ";
   } else {
    debug_msg("category string is undefined");
   }
  }

  my $filter_version = $message->version();
  if (defined $filter_version) {
   $tag = $tag."version=$filter_version ";
  } else {
   debug_msg("filter_version is undefined");
   $tag = $tag."version=0 ";
  }

  my $count = 0;
  $count = $score->getCount();
  if (defined $count) {
   $tag = $tag . "count=$count";
  } else {
  debug_msg("count is undefined");
  }

return $tag;
} # end createXAccessioTag
sub insertXAccessioTag {
```

```
# inserts tag into the MIME header of the message
  my ($message, $tag) = @_;
  $message->addUniqueHeader('X-Accessio-Status', $tag);
} # end of insertXAccessioTag
#
#  for functionality similar to spam assassin
#
sub insertXAccSpamStatusTag {
# inserts tag into the MIME header of the message
  my ($message, $tag) = @_;
  $message->addUniqueHeader('X-Accessio-Spam-Flag', $tag);
} # end of insertXAccessioTag
#
# for spam assassin compatibility
#
sub insertXSpamStatusTag {
# inserts tag into the MIME header of the message
  my ($message, $tag) = @_;
  $message->addUniqueHeader('X-Spam-Status', $tag);
  $message->addUniqueHeader('X-Spam-Flag', $tag);
} # end of insertXAccessioTag
1; #end of MessageTag
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: PipelineEventRegisterSpam.pm,v 1.15 2004/03/29 11:19:10 lizd Exp $
#
#
# this is a derivation of RegisterSpam.pm
# it now checks for a duplicate in the db before
# registering the spam
#
# created 7-24-02 to use LogEvent logging instead of Log
#               added error handling
# modified 8-13-02 by Liz Derr to add removal of forwards
# modified 9-03-02 by Liz Derr to add call for help routine
# modified 11-09-02 by Liz Derr to call Queue Manager for CC
# modified 11-11-02 by LD to write file to dupes first, and then
# rename to nominations if no mathc is found
# modified 3-19-03 more logging to debug apparent "leaks"
# 3-21-03 added moving queue dupes to the queue dupe dir#
# 3-17-04 updated for new pipeline
# 3-17-04 removed eventRegisterSpam()
#############PIPELINE###############
package PipelineEventRegisterSpam;
use strict;
#use MiaVia::WWW::QueueManager;
use Digest::MD5 qw( md5_hex );
use MIME::Parser;
use MiaVia::PipelineSettings;
use MiaVia::PipelineLogEvent;
use MiaVia::Message;
use MiaVia::DbUtil;
use MiaVia::PipelineAccessioBatch;
use MiaVia::Utils;
use MiaVia::PipelineUtils;
use MiaVia::NewMessageDelivery;
use MiaVia::MessageDestination;
##########################################################
# for the new command center, check incoming messages against
# both the spam db and the queued messages db. Insert if brand
# new; queue message appropriately
# called by PipelineQueueMimeSpam
# 11-10-02 by Liz Derr
##########################################################
sub queueSpam {
  my ($entity) = @_;

  my $AccessioConfig= MiaVia::PipelineSettings::getConfig();

  my $hexDig = md5_hex($entity->stringify);
  my $queuedupeDir = $AccessioConfig->queue_dupes();
my $outputDir = $AccessioConfig->CCnom();
my $logDir = $AccessioConfig->log_dir();
my $tmpDir = $AccessioConfig->mime_temp();
my $dupeDir = $AccessioConfig->raw_dupes();
my $bkupDir = $AccessioConfig->forwards();
my $emptyDir = $AccessioConfig->empty_dir();
my $threshold = $AccessioConfig->pass_score();

#my $dbh = MiaVia::DbUtil::openDb('miavia_rdr');

my $fileName = "$dupeDir$hexDig.txt";
my $logString;
my $matchID;
my $filter;
my @scrubbed_msg;
my $line;
my $score = 0;
my $result =0;
my $message; # used for inserting the new queued message
my $message_filename;

# clean up the message - remove forwards, check for dupe
my $head = $entity->head;
my $From = $head->get("From");
# here is where we look for forwards in the message
# body, and remove them if present
my $remove_needed = MiaVia::PipelineUtils::look_for_forwards($entity);
if ($remove_needed eq 'Y') {
  my @newbody = MiaVia::PipelineUtils::remove_forwards($entity->stringify_body, $From);
  # make a backup copy of the original file
  open (OUT, ">$bkupDir$hexDig.txt")
     or warn ( "PipelineEventRegisterSpam: can't open file $bkupDir$hexDig.txt: $!");
  print (OUT $entity->stringify);
  close OUT;
  chmod 0755, "$bkupDir$hexDig.txt";
  # change the message
  push @scrubbed_msg, $entity->stringify_header; #put header first
  push @scrubbed_msg, @newbody; #put body after header
  # note in the log that forwards were removed
  $logString = "Q FORWARDS REMOVED ";
} else {
  push @scrubbed_msg, $entity->stringify;
} # end if remove needed

# write out the message file, with the hex digest file name, into
# the dupe directory
open (OUT, ">$fileName") or
 warn ("PipelineEventRegisterSpam: can't open file $fileName: $!");
foreach my $line (@scrubbed_msg) {
  print (OUT $line);
```

```
}
close OUT;
chmod 0755, "$fileName";

# look for a match in the spam database
($matchID, $filter, $score) = MiaVia::PipelineAccessioBatch::findMatch
            ($hexDig, $dupeDir, "txt",$threshold);
# if match exists, we're done
if ($score >= 0) { #not an empty message
 if ($matchID) { # a match against the spam db
  # no need to rename file since it starts as a spam db duplicate
      $logString = $logString."HANDPRINT DUPLICATE of $matchID scored $score $fileName";
 } else { # not a match against spam db
  my $dbh = MiaVia::PipelineUtils::openQueueDb('miavia_wrtr');
    # look for a match in the queued_messages db, which has both
    # unreviewed messages, problem messages, and non-spam messages
    ($matchID, $message_filename, $score, $message) = MiaVia::PipelineAccessioBatch::findQueuedMatch
             ($hexDig, $dupeDir, "txt",$threshold);
    if ($matchID) {
     # we found a dupe in the queued_messages db, move it to queue dupe dir
     rename $fileName,"$queuedupeDir$hexDig.txt" or
          warn ("can't rename file $fileName to $queuedupeDir$hexDig.txt: $!");
     $fileName = "$queuedupeDir$hexDig.txt"; # for logging purposes
     # increment the SeenCount if Message is still in the QueueHash because
     # not spam and prob messages are still in message db but not in queue
     my $seencount =0;
     $seencount = MiaVia::PipelineUtils::isQueued($matchID,$dbh);
     if ($seencount > 0) { #seencount of 0 means not in QueueHash table
      $seencount++;
      MiaVia::PipelineUtils::updateQueueSeenCount($matchID,$seencount, $dbh);
     }
     $logString = $logString."QDB DUPE of $matchID score: $score $fileName seenCount: $seencount";
     #print STDERR "dupe of Q message found: $matchID seen count $seencount\n";
     } else {
 # a message we've never seen before, so insert it
 # into the queued_messsages db
 rename $fileName,"$outputDir$hexDig.txt" or
        warn ("can't rename file $fileName to $outputDir$hexDig.txt: $!");
       $fileName = "$outputDir$hexDig.txt";
 my $msgDest = new MiaVia::MessageDestination('database',{ dbh => $dbh});
 my $messageID = MiaVia::NewMessageDelivery::deliverMessage($message, $msgDest);
 # insert into QueueHash
 MiaVia::PipelineUtils::insertQueueHash($hexDig,$messageID, $dbh);
     $logString = $logString."Q NEW MESSAGE $messageID score: $score $fileName ";
    }
 } # end if match against spam db
} else { # empty message
 #unlink $fileName;
 rename $fileName,"$emptyDir$hexDig.txt" or
        warn ("can't rename file $fileName to $emptyDir$hexDig.txt: $!");
 $fileName = "$emptyDir$hexDig.txt";
   $logString = $logString."Q EMPTY MESSAGE UNLINKED $fileName ";
  }
  #print STDERR "LOG: $logString\n";

  ##########################################################
  # log the event
  # extract header details for the log using MIME parser
  my $MessageId = $head->get("Message-ID"); # this is the email message id
  my $Sender = $head->get("Sender");
  my $Date = $head->get("Date");
  my $To = $head->get("To");
  my $Subject = $head->get("Subject");
  # Setup the logging
  my $klogger = new PipelineLogEvent(
      TYPE => 'register'
    , LOG_DIR => $logDir
    , FILTER_VERSION => $filter
   , PATH_TO_CRONOLOG => $AccessioConfig->cronopath()
  );
  # now finally log the whole thing
  $klogger->logCCEvent(
          STATUS => "REGISTERQ"
        , COMMENT => $logString
        , SENDER => $Sender
        , FROM => $From
        , TO => $To
        , SUBJECT => $Subject
        , MAIL_MSG_ID => $MessageId # this is the email message id
        , DB_MSG_ID_LIST => $matchID
        , FILENAME => $fileName
        , MAIL_DATE => $Date
       );

  ##########################################################
  # return the result of the queue operation
  return ($result);
}
1; # EventRegisterSpam.pm
```

```
#!/usr/local/bin/perl
# $Id: RevisedInspectionForm.cgi,v 1.96 2004/06/15 02:38:07 lizd Exp $
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2003,2004
#                       All Rights Reserved
#
# Message Inspection and Annotation system:
# allow users to review and annotate the messages in the nominations folder
#
# Liz Derr 11-10-03
# modified 12-13-03 by Liz Derr
# 2-21-04 revised for mingus
# 3-23-04 Revised, new funct for Accessio 6
#
use strict;
use CGI qw/:standard :netscape/;
# protect against DOS attacks
      $CGI::POST_MAX=1024 * 100;  # max 100K posts
      $CGI::DISABLE_UPLOADS = 1;  # no uploads
use MIME::Base64 ();
use MIME::Parser;
use MIME::Head;
use lib "/usr/local/accessio/lib";
use MiaVia::Message;
use MiaVia::InspectionUtils;
use MiaVia::PipelineSettings;
use MiaVia::PipelineUtils;
use MiaVia::DbUtil;
use CGI::MxScreen;
my $fname;
my $q = new CGI;
my $dirname = "/usr/local/accessio/data/"; # a default
my $path = "nominations/";
my $current_file ="";
# start the page
print header;
my $frame_name = path_info();
$frame_name =~ s!^/!!;
# If no path information is provided, then we create
# a side-by-side frame set
if (!$frame_name) {
    print_frameset();
    exit 0;
}
# take action in the various sub-frames as appropriate
print start_html();
get_file()     if $frame_name eq 'file';
#print_finger() if $frame_name eq 'finger';
display_file() if $frame_name eq 'display';
adjust_finger() if $frame_name eq 'adjust';
```

```
change_it() if $frame_name eq 'make_change';
mark_files() if $frame_name eq 'move_it';
get_wild_links() if $frame_name eq 'wild_card';
print end_html();
# Create the frameset - consists of 6 sub-frames, 4 on the left and
# 2 on the right.  The sub-frames on the left offer annotation options
# for the message displayed on the right, in addition to control options
# about which message to display.  The frames on the left display the
# selected message, both in raw form (as it would look in email), and
# in "handprinted" form, as it appears to the Accessio filter, and
# the result of the previous operation.
sub print_frameset {
    my $script = url();
    MiaVia::PipelineSettings->initialize();
    print title('Inspection Frame'),
    frameset({-cols=>'30%,70%'},
          frameset ({-rows=> '25%,20%,45%,10%'},
       frame({-name=>'file',-src=>"$script/file"}),
       frame({-name=>'adjust',-src=>"$script/adjust"}),
       frame({-name=>'wild_card',-src=>"$script/wild_card"}),
       frame({-name=>'make_change',-src=>"$script/make_change"})
      ),
      frameset ({-rows=> '12%,88%'},
         frame({-name=>'mark',-src=>"$script/mark"}),
       frame({-name=>'display',-src=>"$script/display"})
      )
 );
    exit 0;
}
# This frame allows the user to control which message is displayed, and,
# optionally, delete that message from the database.  The user can choose
# to select a random file in the directory of files waiting to be reviewed,
# can select to review a file based on the highest priority of waiting
# messages (at this point determined by the number of copies we've seen
# of this message), or can choose to display a message based on it's
# MD5 file name (the system looks for it in various places), or it's
# message id (if the message is already in the database).
sub get_file {
    my $script = url();
       print start_form(-action=>"$script/display",
    -target=>"display"),
       "File name or a message id ",
       textfield(-name=>'file_name',-size=>35);
       print image_button(-name=>'display_named',
                         -src=>'http://localhost/tools/images/display_named.gif',
                         -value=>'Display Named File');

        print image_button(-name=>'display_msgid',
                         -src=>'http://localhost/tools/images/display_msgid.gif',
                         -value=>'Display Message Id');
        print p();
        print image_button(-name=>'random_inspect',
                        -src=>'http://localhost/tools/images/random_inspection.gif',
                        -value=>'Random Inspection');

        print image_button(-name=>'queue_priority',
                        -src=>'http://localhost/tools/images/queue_priority.gif',
                        -value=>'Queue Priority');

        print p(), hr(), p();
        print image_button(-name=>'delete_named',
                        -src=>'http://localhost/tools/images/delete_file.gif',
                        -value=>'Delete Named File');
        print end_form;
}
# this is the routine that takes action based on the file display
# option selected.  It finds the appropriate file to display, and
# displays the raw and handprinted versions.
sub display_file {
 unless (param) {
        print "No query submitted yet.";
        return;
 }
 my $fname;
 my $score;
 my $message;
 my $matchId;
 my $count;
 my $seen;

 my $action;

 # determine which action to take
 if ( param('display_named.x')) {
  $action = "Display";
 } elsif ( param('random_inspect.x')) {
  $action = "Random";
 } elsif ( param('queue_priority.x')) {
  $action = "Queue";
 } elsif ( param('delete_named.x')) {
  $action = "elete";
 } elsif ( param('display_msgid.x')) {
  $action = "msgid";
 }

      my $dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
      my $queue_dbh = MiaVia::PipelineUtils::openQueueDb('miavia_wrtr');
      my $found_path;

      # determine the path and file name to display based on the action
```

```
if ($action =~ /Display/) {
    $fname = param('file_name');
} elsif ($action =~ /Random/) {
 print p();
    $fname = random_find_file("$dirname$path");
    $found_path="$dirname$path";
} elsif ($action =~ /elete/) {
 $fname = param('file_name');
 if (length($fname) <32) {
  my $temp = MiaVia::DbUtil::getMessage_filename($fname, $dbh);
  $fname = $temp;
 }
} elsif ($action =~ /msgid/) {
 my $msgid = param('file_name');
 $fname = MiaVia::DbUtil::getMessage_filename($msgid, $dbh);
 $found_path = "spam_archive/";
} elsif ($action =~ /Queue/) {
 $fname = MiaVia::PipelineUtils::getMostSeen($queue_dbh);

}
       # take the appropriate action on the specified file
       my $newdirname = $dirname."insp_processed_dupes/";
       $fname =~ s/\s//;
     if (length($fname) > 0) {
      if ($action =~ /elete/) { #delete the message from the db and move the file out of the archive
  MiaVia::DbUtil::deleteMessage_filename($fname, $dbh);
  print p(),"$fname deleted from the spam database";
  my $filename = $dirname."spam_archive/".$fname.".txt";
  my $outfname = $dirname."deleted_spam/".$fname.".txt";
  rename $filename,$outfname;
 } else  { # we shall display a file
  my $dbh = MiaVia::DbUtil::openDb('miavia_rdr');
     if ($action =~ /Random/) { # find a random file in the nomination queue, compuet score
      $fname =~ s/\s//;
   ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$path", $dbh);
   while ( $score >= 40) { # if the score is higher than 40, we consider it a
    # duplicate of a message already in the db, so move it to the dupes
    # directory, and mark the status in the queue db
    # then, go and find the next available file
    my $filename = $dirname.$path.$fname.".txt";
    my $outfname = $newdirname.$fname.".txt";
    rename $filename,$outfname or
     print $q->b("can't rename file $filename to $outfname: $!");
    print p(),"moved $fname with score of $score to processed_dupes",p();
    print p();
    $fname = random_find_file("$dirname$path");
    $fname =~ s/\s//;
     ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$path", $dbh);
   }
   print p();
} elsif ($action =~ /isplay/) {# it display a named file (find path)
 $fname =~ s/\s//;
 $found_path = find_path($fname, $dirname);
 ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$found_path", $dbh);
 $found_path = "$dirname$found_path";
} elsif ($action =~ /ueue/){ # display the next highest priority un-reviewed message
 $found_path = find_path($fname, $dirname);
 while (!(-f "$dirname$found_path$fname.txt")) { #if file not found, try another
  print "unable to find $dirname$found_path$fname, trying a different file",p();
  MiaVia::PipelineUtils::updateQueueHashCategory($fname,"9",$queue_dbh);
  $fname = MiaVia::PipelineUtils::getMostSeen($queue_dbh);
  $found_path = find_path($fname, $dirname);
 }
 ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$found_path", $dbh);
 while ( $score >= 40) { # if the score is higher than 40, we consider it a
  # duplicate of a message already in the db, so move it to the dupes
  # directory, and mark the status in the queue db
  # then, go and find the next available file
  my $filename = $dirname.$found_path.$fname.".txt";
  my $outfname = $newdirname.$fname.".txt";
  rename $filename,$outfname or
   print $q->b("can't rename file $filename to $outfname: $!");
  print p(),"moved $fname with score of $score to processed_dupes",p();
  MiaVia::PipelineUtils::updateQueueHashCategory($fname,"9",$queue_dbh);
  $fname = MiaVia::PipelineUtils::getMostSeen($queue_dbh);
  $found_path = find_path($fname, $dirname);
  while (!(-f "$dirname$found_path$fname.txt")) {
   print "unable to find $fname, trying a different file",p();
   MiaVia::PipelineUtils::updateQueueHashCategory($fname,"9",$queue_dbh);
   $fname = MiaVia::PipelineUtils::getMostSeen($queue_dbh);
   $found_path = find_path($fname, $dirname);
  }
  $fname =~ s/\s//;
   ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$found_path", $dbh);
 }
 $found_path = "$dirname$found_path";
} else {
 ($score,  $matchId, $count, $message) = parse_file($fname, "$dirname$found_path", $dbh);
 $found_path = "$dirname$found_path";
}
# do this for all display options:
# display number of copies seen prior to review
$seen = MiaVia::PipelineUtils::getSeenByName($fname, $queue_dbh);
my $score_string = sprintf("%.2f",$score);
print "$found_path$fname", p(),"score: $score_string matchId: $matchId count $count queue dupes $seen", hr();
mark_options($fname, $found_path, $action); # button bar
# print the email view of the message
print_email($fname, "$found_path");
# print the decomposed handprint view (fingers)
print_fingers($message,$dbh);
```

```
    mark_options($fname, $found_path, $action); # button bar
   }
      } else {
     print p(),"Invalid Filename or Queue is Empty",p();
      }
      if (defined($message)) {
       $message->cleanup();
      }
}
sub print_email {

 # uses the CPAN MIME parser to display the "email view" of the message
 my ($fname, $full_path) = @_;

 my $bh;
 my $parser = new MIME::Parser;
 $parser->output_dir("/usr/local/accessio/data/tmp");
 $parser->tmp_to_core(1);
 $parser->extract_uuencode(1);
 my $entity = $parser->parse_open("$full_path$fname.txt")or die("can't open parser for :$dirname$fname.txt $!");
 print "<p/><P><B>Header information of Original mail message</B></P>";
 print "Subject: ".$entity->get('Subject')."<BR>\n";
 print "Date: ".$entity->get('Date')."<BR>";
 print "From: ".$entity->get('From')."<BR>";
 print "To: ".$entity->get('To')."<BR>";
 print "X-Accessio-Status: ".$entity->get('X-Accessio-Status')."<BR>";
 print "X-Accessio-Recipient: ".$entity->get('X-Accessio-Recipient')."<BR>";

   my $sa_status = $entity->get('X-Spam-Status');
   if ($sa_status =~ /Yes/) {
  print "<H2>(Spam Assassin <B>Yes</B>) X-Spam-Status:</H2> $sa_status<BR>";
 } else {
  print "(Spam Assassin) X-Spam-Status: $sa_status<BR>";
 }

 if ($entity->is_multipart) {
   my @parts = $entity->parts;
   foreach my $part (@parts) {
      if ($bh = $part->bodyhandle) {
       print "<pre>";
         print $bh->as_string;
         print "</pre>";
      }
      if ($part->is_multipart) {
        # nested multipart message
        my @subparts = $part->parts;
        foreach my $subpart (@subparts) {
         if ($bh = $subpart->bodyhandle) {
           print "<pre>";
           print $bh->as_string;
```

```
          print "</pre>";
        } # end if
       } # end for each
     }# end if nested multipart
  } # end foreach
 } else { #isn't multipart
     if ($bh = $entity->bodyhandle) {
      print "<pre>";
        print $bh->as_string;
        print "</pre>";
     } else {
       print "<pre>";
       print $entity->stringify_body;
       print "</pre>";
     }
 } # end else
 $parser->filer->purge;
}
# print the "handprint view" of the message in an HTML table with the digest code
# for each finger being a clickable link that allows the user to take action/annotate
# that particular finger
sub print_fingers {

 my ($message, $dbh) = @_;

 print "<HR>\n";
 my $script = url();
 # setup for html table
 my @headings = ('Count','Weight','Digest Code','Value');
 my @rows = th(\@headings);

 my %para_hash = %{$message->allDigests()};
 my @digest_list = keys %para_hash;
 my $fingerSet = $message->fingerSet();
 my @fingers = MiaVia::PipelineUtils->get_fingers($fingerSet);
 my %value_lookup;
 my $fing_count=0;

 foreach my $finger (@fingers) {
    my $tmp_digest = $finger->digest();
    my $tmp_value = $finger->value();
    $value_lookup{$tmp_digest} = $tmp_value;
    $fing_count++;
 }
 foreach my $digest_code (@digest_list) {
  my $queryStr = qq{ SELECT count(*)
     FROM Finger WHERE Finger_digest_code= };
  $queryStr .= "(\"$digest_code\"";
  $queryStr .= ")";
  my $sth = $dbh->prepare($queryStr);
 $sth->execute();
 my ($finger_count) = $sth->fetchrow_array();
 $sth->finish();
 my $weight = 0;
 if ($finger_count > 0) {
  $weight = MiaVia::PipelineUtils::getWeight($dbh,$digest_code);
 }
 my $tmp_value = $value_lookup{$digest_code};
 if ( MiaVia::DbUtil::is_common_finger($dbh,$digest_code)) {
  push(@rows,td([$finger_count, $weight, submit(-name=>'code_name',-value=>"$digest_code"), i($tmp_value)]));
 } else {
  push(@rows,td([$finger_count, $weight, submit(-name=>'code_name',-value=>"$digest_code"), $tmp_value]));
 }
}
if ($fing_count == 0) {
 print "<p> No Fingers Found for this message</p>";
} else {
 print start_form(-action=>"$script/adjust", -target=>"adjust");
 print $q->hidden(-name=>'value_lookup',-default=>"%value_lookup");
 print table({-border=>undef,},
           caption(b('Finger Counts')),
           Tr(\@rows)
          );

        # display matching wild links, if any
        my @wild_rows = MiaVia::InspectionUtils::wild_link_rows($dbh, $fingerSet);
        if (scalar(@wild_rows) > 0) {
         my @wild_headings = ('WildLink String','Msg Id','Weight');
  my @wild_table_rows = th(\@wild_headings);
         foreach my $wild_row (@wild_rows) {
          my $temp_id = $wild_row->{msgid};
          push (@wild_table_rows,td([$wild_row->{string}, submit(-name=>'msg_id',-value=>"$temp_id"), $wild_row-
>{weight}]));
         }
         print table({-border=>undef,},
             caption(b('Wild Links')),
             Tr(\@wild_table_rows)
           );
        } else {
         print p(),"No Matching Wild Links found for this message";
        }
 print end_form;

}
}
# this is the button bar that allows the user to classify the entire message, as
# spam, not spam, problem, and, if spam, allows the user to select a topical
# classification for the spam message
sub mark_options {
 my $fname = shift;
 my $path = shift;
 my $action = shift;

     my $script = url();

     print $q->start_form( -action=>"$script/move_it", -target=>"mark");
     my $last_file = $q->cookie('filename');
 my $last_dir = $q->cookie('dirname');
 my $last_result = $q->cookie('result');

 print $q->hidden(-name=>'filename',-default=>"$fname");
 print $q->hidden(-name=>'file_action',-default=>"$action");
 print $q->hidden(-name=>'pathname',-default=>"$path");

 print image_button(-name=>'insert',-src=>'http://localhost/tools/images/insert.gif');
        print image_button(-name=>'mark_not',-src=>'http://localhost/tools/images/not_spam.gif');
        print image_button(-name=>'mark_problem',-src=>'http://localhost/tools/images/problem.gif');
        print image_button(-name=>'mark_dupe',-src=>'http://localhost/tools/images/dupe.gif');
        print image_button(-name=>'mark_fpos',-src=>'http://localhost/tools/images/SA_fpos.gif');
        print p();
        print image_button(-name=>'ins_rx',-src=>'http://localhost/tools/images/prescription.jpg');
        print image_button(-name=>'ins_adult',-src=>'http://localhost/tools/images/adult.jpg');
        print image_button(-name=>'ins_credit',-src=>'http://localhost/tools/images/credit.jpg');

        print image_button(-name=>'ins_asian',-src=>'http://localhost/tools/images/asian.jpg');
  print image_button(-name=>'ins_scam',-src=>'http://localhost/tools/images/scam.jpg');

        print end_form;

}
# this routine allows the user to specify an annotation or adjustment to make on a
# specific part of the message (the "finger").  The user can weight the finger, delete
# the finger, and display a list of message filenames that contain the finger
sub adjust_finger {
       my $script = url();
       my $digest = $q->param('code_name');
       if (length($digest) < 2) {
        print p();
       }
       print "$digest";
       print start_form(-action=>"$script/make_change", -target=>"make_change"),p();
       print submit(-name=>'which_action',-value=>'60');
       print submit(-name=>'which_action',-value=>'15');
       print submit(-name=>'which_action',-value=>'1');
       print submit(-name=>'which_action',-value=>'Irrelevant');
       print submit(-name=>'which_action',-value=>'Delete');
       print submit(-name=>'which_action',-value=>'Look Up');
       print submit(-name=>'which_action',-value=>'Make Common');
       print p(),"Wt: ",textfield(-name=>'custom_weight',-size=>6),submit(-name=>'which_action',-value=>'Custom');
```

```
      print $q->hidden(-name=>'digest_code',-default=>"$digest");

      my $msg_id = $q->param('msg_id');

      if (length($msg_id) > 1) {
        print p(),"Message ID: $msg_id",p();
        print p(),"Enter Wild String to Remove: " ,textfield(-name=>'wild_string',-size=>25),p();
        print submit(-name=>'which_action',-value=>'Remove Wild Link');
        print $q->hidden(-name=>'msg_id',-default=>"$msg_id");
       }

      print end_form;

}
# this routine takes the appropriate action on a finger indicated by the user in the
# adjust_finger form
sub change_it {

 unless (param) {
            print "No query submitted yet.";
       return;
   }

 my $dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
 my $digest = $q->param('digest_code');
 my $which_one = $q->param('which_action');
 my $msg_id = $q->param('msg_id');

 if ( (defined($which_one)) && (length($digest) > 1) ) {
  if ($which_one =~ /60/) {
   MiaVia::PipelineUtils::markAllFingersWeight($dbh, $digest,"60");
   print "All fingers with a digest code of $digest have been given a weight of 60",p();
  } elsif ($which_one =~ /15/) {
   MiaVia::PipelineUtils::markAllFingersWeight($dbh, $digest,"15");
   print "All fingers with a digest code of $digest have been given a weight of 15",p();
  } elsif ($which_one =~ /1/) {
   MiaVia::PipelineUtils::markAllFingersWeight($dbh, $digest,"1");
   print "All fingers with a digest code of $digest have been given a weight of 1",p();
  } elsif ($which_one =~ /elete/) {
   MiaVia::PipelineUtils::deleteAllFingers($dbh, $digest);
   print "All fingers with a digest code of $digest have been deleted",p();
  } elsif ($which_one =~ /Irrelevant/) {
   MiaVia::PipelineUtils::markAllFingersAsIrrelevant($dbh, $digest);
   print "All fingers with a digest code of $digest have been marked as irrelevant",p();
  }elsif ($which_one =~ /Custom/) {
   MiaVia::PipelineUtils::markAllFingersAsIrrelevant($dbh, $digest);
   my $custom_weight = $q->param('custom_weight');
   MiaVia::PipelineUtils::markAllFingersWeight($dbh, $digest,$custom_weight);
   print "All fingers with a digest code of $digest have been given a weight of $custom_weight",p();
  }elsif ($which_one =~ /Look/) {
   my @filenames = MiaVia::PipelineUtils::getMessageFilenames($dbh, $digest);
   foreach my $name (@filenames) {
    print p(),"$name";
   }
  }elsif ($which_one =~ /Common/) {
   MiaVia::PipelineUtils::insertCommonFinger($dbh, $digest, "no value available");
   print "The digest code of $digest has been made a Common Finger (not used for lookup or score calc)",p();
  }elsif ($which_one =~ /wild/) {
   insert_wild_links($q);
   my $wild_string = $q->param('wild_string');
   print "WildLink inserted for $wild_string",p();
  }
 } elsif ( (defined($which_one)) && (length($msg_id)> 1) ) {
  if ($which_one =~ /Link/) {
   my $wild_string = $q->param('wild_string');
   MiaVia::PipelineUtils::deleteWildLink($dbh, $msg_id, $wild_string);
   print "Delete wild link $wild_string for Message $msg_id";
  }
 } else {
  print "Error: parameters not defined digest: $digest, which: $which_one, unable to mark fingers",p();
 }
}
# this routine takes the appropriate action on the file indicated by the user in the mark_options
# form
sub mark_files {
 my $mark;
 my $fname = $q->param('filename');
 my $path = $q->param('pathname');

 if ( param('mark_spam.x')) {
  $mark = "As Spam";
 } elsif ( param('mark_not.x')) {
  $mark = "As Not Spam";
 } elsif ( param('mark_problem.x')) {
  $mark = "As Problem";
 } elsif ( param('mark_dupe.x')) {
  $mark = "Dupe";
 } elsif ( param('insert.x')) {
  $mark = "Insert";
 } elsif ( param('mark_fpos.x')) {
  $mark = "SA Fpos";
 } elsif ( param('ins_adult.x')) {
  $mark = "ins_adult";
 } elsif ( param('ins_rx.x')) {
  $mark = "ins_rx";
 } elsif ( param('ins_asian.x')) {
  $mark = "ins_asian";
 } elsif ( param('ins_credit.x')) {
  $mark = "ins_credit";
 } elsif ( param('ins_scam.x')) {
  $mark = "ins_scam";
 }

 my $dbh = MiaVia::DbUtil::openDb('miavia_wrtr');

 if ($mark =~ /Insert/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  print "$fname was inserted.  New Message id is $msgID",p();
 } elsif ($mark =~ /ins_adult/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  MiaVia::PipelineUtils::setCategory($dbh, $msgID,'2');
  print "$fname was inserted. Category is Adult. New Message id is $msgID",p();
 } elsif ($mark =~ /ins_rx/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  MiaVia::PipelineUtils::setCategory($dbh, $msgID,'13');
  print "$fname was inserted. Category is Prescription. New Message id is $msgID",p();
 } elsif ($mark =~ /ins_asian/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  MiaVia::PipelineUtils::setCategory($dbh, $msgID,'15');
  print "$fname was inserted. Category is Asian. New Message id is $msgID",p();
 } elsif ($mark =~ /ins_credit/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  MiaVia::PipelineUtils::setCategory($dbh, $msgID,'12');
  print "$fname was inserted. Category is Credit/Loans. New Message id is $msgID",p();
 } elsif ($mark =~ /ins_scam/) {
  my ($msgID) = insert_message($fname, "$path", $dirname);
  MiaVia::PipelineUtils::setCategory($dbh, $msgID,'14');
  print "$fname was inserted. Category is Scam. New Message id is $msgID",p();
 } else {
        my ($result) = move_files ($mark, $fname, "$path", $dirname);
        print "$fname $result",p();
 }


}
# this routine inserts or deletes a wild link based on the action specified by the user
# in the get_wild_links frame
sub insert_wild_links {

 my $q = shift;

 my $msg_id = "";
 my $fname = "";
 my $wild_string = "";
 my $sublink = "";
 my @subLinks=("");
 my $weight="1";
 my $name;

 $fname = $q->param('filename');
 $msg_id = $q->param('msg_id');
 $sublink = $q->param('sublink');
 $weight = $q->param('weight');
 $wild_string = $q->param('wild_string');

 @subLinks=("$sublink");

 my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');

 # if a name has been specified instead of a message id, get the message id
 if (
  (length($fname) > 1)
  &&  ( ($msg_id =~ /msg id here/) || (length($msg_id) < 1))
  ) {
  $msg_id = MiaVia::DbUtil::getMessage_id($fname, $out_dbh);
 }
 MiaVia::DbUtil::insertWildSubs($out_dbh, $msg_id, $wild_string, \@subLinks, $weight);
}
# this frame allows the user to enter a wild-card link for the message by specifying the
# message id (the message must first be inserted into the database), the wild card string,
# and the sub-link that will be used to lookup the wildcard string in the database
sub get_wild_links {
 my $msg_id = "msg id here";
 my $fname = "or enter filename with no .txt here";
 my $wild_string = "wild string here";
 my $sublink = "sublink here";
 my $weight="1000";

 my $script = url();

 print start_form(-action=>"$script/make_change", -target=>"make_change"),p();

 # get message id
 print p(),"Message id (must be in db): ";
 print $q->textfield(-name => 'msg_id', -label =>'Message Id: ', -value => $msg_id, -size =>20);
 # get wild string
 print p(),"Enter the wild card string: ";
 print $q->textfield(-name => 'wild_string', -label =>'REQUIRED - Wildcard String: ', -value => $wild_string, -size =>30);
 # get sublink for the lookup
 print p(),"Enter the sub link: ";
 print $q->textfield(-name => 'sublink', -label =>'REQUIRED - Sublink: ', -value => $sublink, -size =>30);
 # get weight
 print p(),"Enter the weight: ";
 print $q->textfield(-name => 'weight', -label =>'Weight: ', -value => $weight, -size =>10);
 print p($q->submit( -value => "submit"));

 print $q->hidden(-name=>'digest_code',-default=>"none");
 print $q->hidden(-name=>'which_action',-default=>"wild");
}
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: Utils.pm,v 1.84 2004/03/05 09:11:52 lizd Exp $
#
#
# these are general utilities
#
# created 8-07-02 by Liz Derr
# functions added 6-09-03 by Erik Kargard
# Removed References to MiaVia::Settings (mostly debugs), and obsolete get_dbversion,
# also removed use DbUtil;
# 2-27-04 added facility and priority used by SysLogEvent
package MiaVia::Utils;
use MIME::Parser;
# core settings
our $pass_score;  # spam threshold
our $stop_looking; # score at which to stop looking for matches
our $x_threshold_tag; # whether or not to put threshold into X-header tag
our $categorize; # whether or not to put category into X-header tag
our $short_stop; # stop looking score for short messages
our $short_pass; #spam threshold for short messages
our $short_message; # threshold size of message to be considered short
our $special_short; # whether or not to treat short messages specially
our $mime_temp;
our $mysql_servername;
our $facility;
our $priority;
sub set_pass_score {
 my $self = shift;
 my $tmp = shift;
 $pass_score = $tmp;
}
sub set_stop_looking {
 my $self = shift;
 my $tmp = shift;
 $stop_looking = $tmp;
}
sub set_x_threshold_tag {
 my $self = shift;
 my $tmp = shift;
 $x_threshold_tag = $tmp;
}
sub set_categorize {
 my $self = shift;
 my $tmp = shift;
 $categorize = $tmp;
}
sub set_short_stop {
 my $self = shift;
 my $tmp = shift;
 $short_stop = $tmp;
}
sub set_short_pass {
 my $self = shift;
 my $tmp = shift;
 $short_pass = $tmp;
}
sub set_short_message {
 my $self = shift;
 my $tmp = shift;
 $short_message = $tmp;
}
sub set_special_short {
 my $self = shift;
 my $tmp = shift;
 $special_short = $tmp;
}
sub set_mime_temp {
 my $self = shift;
 my $tmp = shift;
 $mime_temp = $tmp;
}
sub set_mysql_servername {
 my $self = shift;
 my $tmp = shift;
 $mysql_servername = $tmp;
}
sub set_facility {
 my $self = shift;
 my $tmp = shift;
 $facility = $tmp;
}
sub set_priority {
 my $self = shift;
 my $tmp = shift;
 $priority = $tmp;
}
sub get_pass_score {
 return $pass_score;
}
sub get_stop_looking {
 return $stop_looking;
}
sub get_x_threshold_tag {
 return $x_threshold_tag;
}
sub get_categorize {
 return $categorize;
}
```

```
sub get_short_stop {
 return $short_stop;
}
sub get_short_pass {
 return $short_pass;
}
sub get_short_message {
 return $short_message;
}
sub get_special_short {
 return $special_short;
}
sub get_mime_temp {
 return $mime_temp;
}
sub get_mysql_servername {
 return $mysql_servername;
}
sub get_facility {
 return $facility;
}
sub get_priority {
 return $priority;
}
# returns the age in seconds of the specified file, and the size in bytes
sub file_info {
    my $filename = shift;
    # print "filename is $filename\n";
    my ($dev, $ino,$mode,$nlink, $uid, $gid, $rdev, $size,
    $atime, $mtime, $ctime, $blksize, $blocks)  = stat $filename;
    my $age = time() - $mtime;
    #print "dev=$dev, ino=$ino, mode=$mode, nlink=$nlink, uid=$uid, " .
    #     "gid=$gid, rdev=$rdev, size=$size, atime=$atime, mtime=$mtime, " .
    #     "ctime=$ctime, blksize=$blksize, blocks=$blocks\n";
    # print "age is $age seconds";
    return $age, $size;
}
# returns the year and year day of the specified file
sub file_date {
    my $filename = shift;
    my ($dev, $ino,$mode,$nlink, $uid, $gid, $rdev, $size,
    $atime, $mtime, $ctime, $blksize, $blocks)  = stat $filename;
    my ($sec, $min, $hour, $dom, $mon, $year, $wday, $yday, $isdst) =
     localtime($mtime);
    return $year+1900, $yday;
}
#
#
sub alive {
 print "I'm alive!\n";
}
sub call_for_help {
  my $subject = shift;
  my $errorStr = shift;
  warn "Error: $errorStr\n";
}
# Usage: normalize_url ($url_string)
#
#       url_string a URL
#               Assumes that $url_string has been pre-processed
#    stripped http://, @ before host, cgi args, #'s, etc.
#               and is in the format:
#                       host/resource
#               Host is DNS name or ip in dot notation, dword,
#               octal, hex or combination format
#               Resource can be text or encoded ascii characters
#       Returns $string
#  Normalized as best as could
#  Host - text or dot ip notation
#  Resource - Hex encoding converted to ascii
#
# If passed $url_string is invalid the normalization will be attempted
# hopefully fail and passed $url_string returned.
#
# ONLY DECODES HEX
# HEX decode should be separate function
# decode done separate from remaing URL normalization
# Assumes single line
# Does not check input string
# Does not check output string and remove non needed URL parts
sub normalize_url {
        my $src = shift;
 if (!defined($src)) {
  warn "no source string passed to normalize_url\n";
 }
        # Decode URL-encoded stuff
 # Only decode safe chars
        #$src =~  s/%((2[ef]|3[0-9a]|4[0-9a-f]|5[0-9a]|6[1-9a-f]|7[0-9a]))/chr(hex("0x$2"))/gsei;
 # Decode printable - chars 0x20 " " - 0x7E "~"
        $src =~ s/%((2[0-9a-f]|[2-6][0-9a-f]|7[0-9a-e]))/chr(hex("0x$2"))/gsei;
        return $src;
}
#
# Usage: strip_html_noise ($html_string)
# Strips noise tags from HTML
#       "<!....>" strings are deleted
#
#       html_string - passed HTML string
#       Returns $string with all noise ("<!....>") deleted
#
```

```
# If passed $html_string is invalid the stripping will be attempted
# hopefully fail and passed $html_string returned.
#
sub strip_html_noise {
 my $src = shift;
 if (!defined($src)) {
  warn "no source string passed to strip_html_noise\n";
 }
 # pattern: <! - all non-special characters - >
 #s/<![-\s]*([\sa-zA-Z0-9])*[-\s]*>//gxso;
 # pattern: <!-- match ws and alphanum end with >
 #s/<!--([\sa-zA-Z0-9])*[-\s]*>//gxso;
 # pattern: <! - non-special characters and "@" "."- >
 $src =~ s/<![-\s]*([\sa-zA-Z0-9@.])*[-\s!]*>//gxso;
 return $src;
}

sub decode_hex {
 my $src = shift;
 if (defined($src)) {
  $src =~ s/%((2[0-9a-f]|[2-6][0-9a-f]|7[0-9a-e]))/chr(hex("0x$2"))/gsei;
 } else {
  warn "no source string passed to decode_hex\n";
 }
 return $src;
}
1;
```

SYSTEM AND METHOD FOR ASSOCIATING STRUCTURED AND MANUALLY SELECTED ANNOTATIONS WITH ELECTRONIC DOCUMENT CONTENTS

Instructions for Installing and Operating the Message Inspection/Annotation Program.

This package consists of series of program listing files, each of which should be placed within one of the following directories:

bin, which contains perl scripts run primarily for reporting purposes;

cgi, which contains the web-based UI scripts for the system;

sql, which contains the database table definitions;

images, which contains the images used by the cgi scripts for the UI;

sbin, which contains shell scripts that run the reporting scripts; and

lib/MiaVia, which contains the accessio library files that contain the "business logic" of the system (i.e. the program code other than the pure UI).

The files are to be installed as described below along with supporting software modules, also described below.

The inventive software program files along with the supporting software modules are to be installed on a server computer device of a type widely known to those skilled in the art. The server computer must conform, at least, to the following minimum system requirements:

Processor: Pentium 4 (2Ghz)

Memory: 1 GB RAM (64-128 MB available)

Disk space: 1 GB free disk space minimum

Operating system: Linux Red Hat 7.3, 8.0, 9.0 or Enterprise Server

The program files should be placed in the following named directories after undertaking the installation of the supporting software modules. Each file has been submitted as an attachment to the EPAVE electronic submission to the USPTO with a .txt file extension. It is necessary to rename each file according to the file extensions listed below in order for each file to function properly. Files listed under the "images" directory below have not been submitted. These files may be created by the user as .gif or .jpg image files as listed below. The image files serve as graphical user interface buttons, each of which should display a graphically-rendered word that matches the file name. The particular style of graphic rendering is insignificant as long as the user can determine the button function from the word displayed in each graphical image.

bin:

BRegSFile.pl

FindUnmarkedMessagesInRegLogs.pl

InsertCount.pl

MimeScrubFiles.pl

RemoveNonSpam1.pl

RemoveNonSpam2.pl

RemoveNonSpam3.pl

RemoveNonSpam4.pl

RemoveProbableNonSpam.pl

RemoveUncatchableSpam.pl

ReportFpos.pl

cgi:

MingScrubNoms.cgi

RevisedInspectionForm.cgi

images:

adult.jpg

asian.jpg

credit.jpg

delete_file.gif

display_msgid.gif

display_named.gif

dupe.gif
not_spam.gif
prescription.jpg
problem.gif
queue_priority.gif
random_inspection.gif
SA_fpos.gif
scam.gif
spam.gif
lib\MiaVia:
DbUtil.pm
FingerSpec.pm
HtmlTagProcessor.pm
InspectionUtils.pm
Message.pm
MessageDestination.pm
MessageSource.pm
MessageTag.pm
NewFinger.pm
NewFingerSet.pm
NewHtmlBodyProcessor.pm
NewMatch.pm
NewMessageDelivery.pm
NewMessageFactory.pm
NewMimeProcessor.pm
NewScore.pm
NewTextBodyProcessor.pm
PipelineEventRegisterSpam.pm
PipelineLogEvent.pm
PipelineQueueMimeSpam.pm
PipelineRegisterSpam.pm
PipelineSettings.pm
PipelineUtils.pm
SubLink.pm
SysLogEvent.pm
TextParagraph.pm
TextParagraph.pm
sbin:
FposSummaryReport.sh
InsertReport.sh
RegistrationReport.sh
ReportFalsePositives.sh
ScrubFilesToInspect.sh
sql:
CommonFingers.sql
CommonSubLinks.sql
new_create_dbs.sql
new_toaster_db_setup.sql
NewLinkTables.sql
queue_msg_db.sql

This system currently runs on a Debian (Woody) linux server running MySQL 3.23.58, perl 5.61 (configuration details below), Apache 1.3 web server (configuration details below), and requires the CPAN perl modules listed below.
To install and run this software:
First, use the code in the sql directory to create the queued_messages and accessio_handprints databases. Create the following users: miavia_rdr and miavia_wrtr, with the sql provided in new_create_dbs.sql and supply them with passwords (replace the "XXXXX" defaults). Put these passwords into the lib/MiaVia/DbUtil.pm file (replace the "XXXXX" defaults).
Second create the image, cgi, and lib/MiaVia folders in a place that your web server can find them, and copy the files from the folders in this package to these folders. The default path for the lib files is /usr/local/accessio/lib/MiaVia. The default path for the cgi and image dirs is /home/web/tools.
Third, get a directory of sample email messages. The default directory for the nominations queue is /usr/local/accessio/data/nominations. If you don't install with these defaults, then you'll need to change the path strings in the cgi code appropriately to find the files you're looking for.
From this you should be able to run the inspection UI. The reporting and scrubbing scripts are included for reference, but aren't required to run the basic UI.
The main inspection code is in RevisedInspectionForm.cgi. MingScrubNoms.cgi
Here is the configration information for Apache:

```
Server version: Apache/1.3.26 (Unix) Debian GNU/Linux
Server built:   Oct 26 2002 09:15:15
Server's Module Magic Number: 19990320:13
Server compiled with....
 -D EAPI
 -D HAVE_MMAP
 -D HAVE_SHMGET
 -D USE_SHMGET_SCOREBOARD
 -D USE_MMAP_FILES
 -D HAVE_FCNTL_SERIALIZED_ACCEPT
 -D HAVE_SYSVSEM_SERIALIZED_ACCEPT
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D HTTPD_ROOT="/usr"
 -D SUEXEC_BIN="/usr/lib/apache/suexec"
 -D DEFAULT_PIDLOG="/var/run/apache.pid"
 -D DEFAULT_SCOREBOARD="/var/run/apache.scoreboard"
 -D DEFAULT_LOCKFILE="/var/run/apache.lock"
 -D DEFAULT_ERRORLOG="/var/log/apache/error.log"
 -D TYPES_CONFIG_FILE="/etc/mime.types"
 -D SERVER_CONFIG_FILE="/etc/apache/httpd.conf"
 -D ACCESS_CONFIG_FILE="/etc/apache/access.conf"
 -D RESOURCE_CONFIG_FILE="/etc/apache/srm.conf"
```

Here is the perl configuration information:

```
 Summary of my perl5 (revision 5.0 version 6 subversion 1) configuration:
  Platform:
    osname=linux, osvers=2.4.20-7um, archname=i386-linux
    uname='linux (none) 2.4.20-7um #1 smp fri aug 8 18:30:28 edt 2003 i686 unknown '
    config_args='-Dccflags=-DDEBIAN -Dcccdlflags=-fPIC -Darchname=i386-linux -Dprefix=/usr -
Dprivlib=/usr/share/perl/5.6.1 -Darchlib=/usr/lib/perl/5.6.1 -Dvendorprefix=/usr -Dvendorlib=/usr/share/perl5 -
```

```
Dvendorarch=/usr/lib/perl5 -Dsiteprefix=/usr/local -Dsitelib=/usr/local/share/perl/5.6.1 -
Dsitearch=/usr/local/lib/perl/5.6.1 -Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3 -Dman1ext=1 -
Dman3ext=3perl -Dpager=/usr/bin/sensible-pager -Uafs -Ud_csh -Uusesfio -Duseshrplib -Dlibperl=libperl.so.5.6.1 -
Dd_dosuid -des'
    hint=recommended, useposix=true, d_sigaction=define
    usethreads=undef use5005threads=undef useithreads=undef usemultiplicity=undef
    useperlio=undef d_sfio=undef uselargefiles=define usesocks=undef
    use64bitint=undef use64bitall=undef uselongdouble=undef
  Compiler:
    cc='cc', ccflags ='-DDEBIAN -fno-strict-aliasing -I/usr/local/include -D_LARGEFILE_SOURCE -
D_FILE_OFFSET_BITS=64',
    optimize='-O2',
    cppflags='-DDEBIAN -fno-strict-aliasing -I/usr/local/include'
    ccversion='', gccversion='2.95.4 20011002 (Debian prerelease)', gccosandvers=''
    intsize=4, longsize=4, ptrsize=4, doublesize=8, byteorder=1234
    d_longlong=define, longlongsize=8, d_longdbl=define, longdblsize=12
    ivtype='long', ivsize=4, nvtype='double', nvsize=8, Off_t='off_t', lseeksize=8
    alignbytes=4, usemymalloc=n, prototype=define
  Linker and Libraries:
    ld='cc', ldflags =' -L/usr/local/lib'
    libpth=/usr/local/lib /lib /usr/lib
    libs=-lgdbm -ldb -ldl -lm -lc -lcrypt
    perllibs=-ldl -lm -lc -lcrypt
    libc=/lib/libc-2.2.5.so, so=so, useshrplib=true, libperl=libperl.so.5.6.1
  Dynamic Linking:
    dlsrc=dl_dlopen.xs, dlext=so, d_dlsymun=undef, ccdlflags='-rdynamic'
    cccdlflags='-fPIC', lddlflags='-shared -L/usr/local/lib'
Characteristics of this binary (from libperl):
  Compile-time options: USE_LARGE_FILES
  Built under linux
  Compiled at Jan 11 2002 04:09:18
  %ENV:
    PERL5LIB="/usr/local/accessio/lib"
  @INC:
    /usr/local/accessio/lib
    /usr/local/lib/perl/5.6.1
    /usr/local/share/perl/5.6.1
    /usr/lib/perl5
    /usr/share/perl5
    /usr/lib/perl/5.6.1
    /usr/share/perl/5.6.1
    /usr/local/lib/site_perl
    .
Module Paths /usr/local/accessio/lib
/usr/local/lib/perl/5.6.1
/usr/local/share/perl/5.6.1
/usr/lib/perl5
/usr/share/perl5
/usr/lib/perl/5.6.1
/usr/share/perl/5.6.1
```

```
/usr/local/lib/site_perl
.

Path(s) to TAR tar:
/bin/tar
/usr/include/tar.h
/usr/share/man/man1/tar.1.gz
Path(s) to GZIP gzip:
/bin/gzip
/usr/share/man/man1/gzip.1.gz
Path to APACHE/HTTPD apache:
/usr/sbin/apache
/etc/apache.bak
/etc/apache
/usr/lib/apache
/usr/share/apache
/usr/share/man/man8/apache.8.gz
httpd:

Path to PHP php:
Path to MYSQL mysql:
/usr/bin/mysql
/etc/mysql
/usr/local/bin/mysql
/usr/local/mysql
/usr/share/mysql
/usr/share/man/man1/mysql.1.gz
Path to MAN (Unix manual) man:
/usr/bin/man
/usr/local/man
/usr/share/man
/usr/share/man/man1/man.1.gz
/usr/share/man/man7/man.7.gz
Path to PERLDOC perldoc:
/usr/bin/perldoc


Here are the installed perl modules:
1. AnyDBM_File
2. AppConfig
3. AppConfig::Args
4. AppConfig::CGI
5. AppConfig::File
6. AppConfig::Getopt
7. AppConfig::State
8. AppConfig::Sys
9. Archive::Tar
10. Archive::Tar::Constant
11. Archive::Tar::File
12. attributes
```

13. attrs
14. AutoLoader
15. AutoSplit
16. autouse
17. B
18. B::Asmdata
19. B::Assembler
20. B::Bblock
21. B::Bytecode
22. B::C::Section
23. B::CC
24. B::Concise
25. B::Debug
26. B::Deparse
27. B::Disassembler::BytecodeStream
28. B::Lint
29. B::Showlex
30. B::Stackobj
31. B::Stash
32. B::Terse
33. B::Xref
34. base
35. Benchmark
36. blib
37. Bundle::CGI::Builder::Complete
38. Bundle::CPAN
39. Bundle::DBD::mysql
40. Bundle::DBI
41. Bundle::LWP
42. ByteLoader
43. bytes
44. Carp
45. Carp::Datum
46. Carp::Datum::Assert
47. Carp::Datum::Cfg
48. Carp::Datum::Flags
49. Carp::Datum::MakeMaker
50. Carp::Datum::Strip
51. CGI
52. CGI::Carp
53. CGI::Cookie
54. CGI::MxScreen
55. CGI::MxScreen::Action_Env
56. CGI::MxScreen::Config
57. CGI::MxScreen::Constant
58. CGI::MxScreen::Error
59. CGI::MxScreen::Exception
60. CGI::MxScreen::Exception::Bounce
61. CGI::MxScreen::Form::Button
62. CGI::MxScreen::Form::Field

63. CGI::MxScreen::Form::Utils
64. CGI::MxScreen::HTML
65. CGI::MxScreen::Layout
66. CGI::MxScreen::Screen
67. CGI::MxScreen::Serializer
68. CGI::MxScreen::Serializer::Storable
69. CGI::MxScreen::Session
70. CGI::MxScreen::Session::Medium
71. CGI::MxScreen::Session::Medium::Browser
72. CGI::MxScreen::Session::Medium::File
73. CGI::MxScreen::Session::Medium::Raw_File
74. CGI::MxScreen::Tie::Buffered_Output
75. CGI::MxScreen::Tie::Read_Checked
76. CGI::MxScreen::Tie::Sinkable
77. CGI::MxScreen::Tie::Stdout
78. CGI::Persistent
79. CGI::Pretty
80. CGI::Push
81. CGI::TabPane
82. CGI::Test
83. CGI::Test::Form
84. CGI::Test::Form::Group
85. CGI::Test::Form::Widget
86. CGI::Test::Form::Widget::Box
87. CGI::Test::Form::Widget::Box::Check
88. CGI::Test::Form::Widget::Box::Radio
89. CGI::Test::Form::Widget::Button
90. CGI::Test::Form::Widget::Button::Image
91. CGI::Test::Form::Widget::Button::Plain
92. CGI::Test::Form::Widget::Button::Reset
93. CGI::Test::Form::Widget::Button::Submit
94. CGI::Test::Form::Widget::Hidden
95. CGI::Test::Form::Widget::Input
96. CGI::Test::Form::Widget::Input::File
97. CGI::Test::Form::Widget::Input::Password
98. CGI::Test::Form::Widget::Input::Text_Area
99. CGI::Test::Form::Widget::Input::Text_Field
100. CGI::Test::Form::Widget::Menu
101. CGI::Test::Form::Widget::Menu::List
102. CGI::Test::Form::Widget::Menu::Popup
103. CGI::Test::Input
104. CGI::Test::Input::Multipart
105. CGI::Test::Input::URL
106. CGI::Test::Page
107. CGI::Test::Page::Error
108. CGI::Test::Page::HTML
109. CGI::Test::Page::Other
110. CGI::Test::Page::Real
111. CGI::Test::Page::Text
112. CGI::Util

113. charnames
114. Class::Struct
115. Compress::Zlib
116. Config
117. constant
118. CPAN
119. CPAN::Admin
120. CPAN::Mirrored::By
121. CPAN::Nox
122. Crypt::Blowfish
123. Crypt::CBC
124. Cwd
125. Data::Dumper
126. DB
127. DBD::ExampleP
128. DBD::mysql
129. DBD::NullP
130. DBD::Proxy
131. DBD::Sponge
132. DBI
133. DBI::Const::GetInfo::ANSI
134. DBI::Const::GetInfo::ODBC
135. DBI::Const::GetInfoReturn
136. DBI::Const::GetInfoType
137. DBI::FAQ
138. DBI::Format
139. DBI::Profile
140. DBI::ProxyServer
141. DBI::Shell
142. Debconf::AutoSelect
143. Debconf::Base
144. Debconf::Client::ConfModule
145. Debconf::Config
146. Debconf::ConfModule
147. Debconf::Db
148. Debconf::DbDriver
149. Debconf::DbDriver::Backup
150. Debconf::DbDriver::Cache
151. Debconf::DbDriver::Copy
152. Debconf::DbDriver::Debug
153. Debconf::DbDriver::Directory
154. Debconf::DbDriver::DirTree
155. Debconf::DbDriver::File
156. Debconf::DbDriver::LDAP
157. Debconf::DbDriver::Pipe
158. Debconf::DbDriver::Stack
159. Debconf::Element
160. Debconf::Element::Dialog::Boolean
161. Debconf::Element::Dialog::Multiselect
162. Debconf::Element::Dialog::Note

163. Debconf::Element::Dialog::Password
164. Debconf::Element::Dialog::Select
165. Debconf::Element::Dialog::String
166. Debconf::Element::Dialog::Text
167. Debconf::Element::Editor::Boolean
168. Debconf::Element::Editor::Multiselect
169. Debconf::Element::Editor::Note
170. Debconf::Element::Editor::Password
171. Debconf::Element::Editor::Select
172. Debconf::Element::Editor::String
173. Debconf::Element::Editor::Text
174. Debconf::Element::Gnome
175. Debconf::Element::Gnome::Boolean
176. Debconf::Element::Gnome::Multiselect
177. Debconf::Element::Gnome::Note
178. Debconf::Element::Gnome::Password
179. Debconf::Element::Gnome::Select
180. Debconf::Element::Gnome::String
181. Debconf::Element::Gnome::Text
182. Debconf::Element::Multiselect
183. Debconf::Element::Noninteractive
184. Debconf::Element::Noninteractive::Note
185. Debconf::Element::Noninteractive::Select
186. Debconf::Element::Select
187. Debconf::Element::Teletype::Boolean
188. Debconf::Element::Teletype::Multiselect
189. Debconf::Element::Teletype::Note
190. Debconf::Element::Teletype::Password
191. Debconf::Element::Teletype::Select
192. Debconf::Element::Teletype::String
193. Debconf::Element::Teletype::Text
194. Debconf::Element::Web::Boolean
195. Debconf::Element::Web::Multiselect
196. Debconf::Element::Web::Note
197. Debconf::Element::Web::Password
198. Debconf::Element::Web::Select
199. Debconf::Element::Web::String
200. Debconf::Element::Web::Text
201. Debconf::Format
202. Debconf::Format::822
203. Debconf::FrontEnd
204. Debconf::FrontEnd::Dialog
205. Debconf::FrontEnd::Editor
206. Debconf::FrontEnd::Gnome
207. Debconf::FrontEnd::Noninteractive
208. Debconf::FrontEnd::Passthrough
209. Debconf::FrontEnd::Readline
210. Debconf::FrontEnd::ScreenSize
211. Debconf::FrontEnd::Teletype
212. Debconf::FrontEnd::Text

213. Debconf::FrontEnd::Web
214. Debconf::Gettext
215. Debconf::Iterator
216. Debconf::Log
217. Debconf::Priority
218. Debconf::Question
219. Debconf::Template
220. Debconf::Template::Transient
221. Debconf::TmpFile
222. Debian::DebConf::Client::ConfModule
223. DebianNet
224. Devel::AutoProfiler
225. Devel::DProf
226. Devel::LineTrace
227. Devel::OpProf
228. Devel::Peek
229. Devel::SelfStubber
230. MiaVia::SubLink
231. MiaVia::SysLogEvent
232. MiaVia::TextParagraph
233. MiaVia::Utils
234. MiaVia::ViewHtmlBodyProcessor
235. MiaVia::ViewHtmlTagProcessor
236. MiaVia::ViewMessageFactory
237. MiaVia::ViewMimeProcessor
238. MiaVia::ViewTextBodyProcessor
239. MiaVia::WWW
240. MIME::Base64
241. MIME::Body
242. MIME::Decoder
243. MIME::Decoder::Base64
244. MIME::Decoder::Binary
245. MIME::Decoder::Gzip64
246. MIME::Decoder::NBit
247. MIME::Decoder::QuotedPrint
248. MIME::Decoder::UU
249. MIME::Entity
250. MIME::Field::ContDisp
251. MIME::Field::ConTraEnc
252. MIME::Field::ContType
253. MIME::Field::ParamVal
254. MIME::Head
255. MIME::Parser
256. MIME::Parser::Filer
257. MIME::Parser::Reader
258. MIME::Parser::Results
259. MIME::QuotedPrint
260. MIME::Tools
261. MIME::WordDecoder
262. MIME::Words

263. Module::Build
264. Module::Build::Base
265. Module::Build::Compat
266. Module::Build::Cookbook
267. Module::Build::Platform::aix
268. Module::Build::Platform::Amiga
269. Module::Build::Platform::cygwin
270. Module::Build::Platform::darwin
271. Module::Build::Platform::Default
272. Module::Build::Platform::EBCDIC
273. Module::Build::Platform::MacOS
274. Module::Build::Platform::MPEiX
275. Module::Build::Platform::os2
276. Module::Build::Platform::RiscOS
277. Module::Build::Platform::Unix
278. Module::Build::Platform::VMS
279. Module::Build::Platform::VOS
280. Module::Build::Platform::Windows
281. Module::Build::PodParser
282. Module::Build::PPMMaker
283. MY
284. My::WebApp
285. MyClass
286. Mysql
287. Mysql::Statement
288. MyTimer
289. NDBM_File
290. Net::Cmd
291. Net::Config
292. Net::Domain
293. Net::FTP
294. Net::FTP::A
295. Net::FTP::dataconn
296. Net::FTP::E
297. Net::FTP::I
298. Net::FTP::L
299. Net::hostent
300. Net::HTTP
301. Net::HTTP::Methods
302. Net::HTTP::NB
303. Net::HTTPS
304. Net::netent
305. Net::Netrc
306. Net::NNTP
307. Net::Ping
308. Net::POP3
309. Net::protoent
310. Net::servent
311. Net::SMTP
312. Net::Telnet

313. Net::Time
314. O
315. ODBM_File
316. Opcode
317. open
318. ops
319. overload
320. Parser
321. PipelineEventRegisterSpam
322. PipelineLogEvent
323. Pod::Checker
324. Pod::Find
325. Pod::Functions
326. Pod::Html
327. Pod::InputObjects
328. Pod::LaTeX
329. Pod::Man
330. Pod::ParseLink
331. Pod::Parser
332. Pod::ParseUtils
333. Pod::Plainer
334. Pod::Select
335. Pod::Text
336. Pod::Text::Color
337. Pod::Text::Overstrike
338. Pod::Text::Termcap
339. Pod::Usage
340. POSIX
341. re
342. readline
343. RegisterSpam
344. Safe
345. SDBM_File
346. Search::Dict
347. SelectSaver
348. SelfLoader
349. Shell
350. sigtrap
351. Socket
352. Storable
353. strict
354. subs
355. Symbol
356. Sys::Hostname
357. Sys::Syslog
358. Term::ANSIColor
359. Term::Cap
360. Term::Complete
361. Term::ReadKey
362. Term::ReadLine::Perl

363. Term::ReadLine::Stub
364. Test
365. Test::Builder
366. Test::Harness
367. Test::Harness::Assert
368. Test::Harness::Iterator
369. Test::Harness::Straps
370. Test::More
371. Test::Simple
372. Text::Abbrev
373. Text::ParseWords
374. Text::Soundex
375. Text::Tabs
376. Text::Wrap
377. Tie::Array
378. Tie::Handle
379. Tie::Hash
380. Tie::RefHash
381. Tie::Scalar
382. Tie::SubstrHash
383. Time::gmtime
384. Time::HiRes
385. Time::Local
386. Time::localtime
387. Time::tm
388. UNIVERSAL
389. URI
390. URI::_foreign
391. URI::_generic
392. URI::_login
393. URI::_query
394. URI::_segment
395. URI::_server
396. URI::_userpass
397. URI::data
398. URI::Escape
399. URI::file
400. URI::file::Base
401. URI::file::FAT
402. URI::file::Mac
403. URI::file::OS2
404. URI::file::QNX
405. URI::file::Unix
406. URI::file::Win32
407. URI::ftp
408. URI::gopher
409. URI::Heuristic
410. URI::http
411. URI::https
412. URI::ldap

413. URI::mailto
414. URI::news
415. URI::nntp
416. URI::pop
417. URI::rlogin
418. URI::rsync
419. URI::rtsp
420. URI::rtspu
421. URI::snews
422. URI::ssh
423. URI::telnet
424. URI::URL
425. URI::urn
426. URI::urn::isbn
427. URI::urn::oid
428. URI::WithBase
429. User::grent
430. User::pwent
431. utf8
432. vars
433. warnings
434. WWW::RobotRules
435. WWW::RobotRules::AnyDBM_File
436. XML::LibXML
437. XML::LibXML::Boolean
438. XML::LibXML::Literal
439. XML::LibXML::NodeList
440. XML::LibXML::Number
441. XML::LibXML::SAX::Builder
442. XML::LibXML::SAX::Generator
443. XML::LibXML::SAX::Parser
444. XML::NamespaceSupport
445. XML::SAX
446. XML::SAX::Base
447. XML::SAX::DocumentLocator
448. XML::SAX::Exception
449. XML::SAX::ParserFactory
450. XML::SAX::PurePerl
451. XML::SAX::PurePerl::DebugHandler
452. XML::SAX::PurePerl::Exception
453. XML::SAX::PurePerl::Productions
454. XML::SAX::PurePerl::Reader
455. XML::SAX::PurePerl::Reader::Stream
456. XML::SAX::PurePerl::Reader::String
457. XML::SAX::PurePerl::Reader::URI
458. XSLoader

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# $Id: MessageDestination.pm,v 1.12 2004/06/04 02:26:28 lizd Exp $
# clear of Settings and Util references
#
# 6-3-04 removed XML
package MiaVia::MessageDestination;
use strict;
#use XML::LibXML;
## CLASS METHODS
##
# Construct a new MessageDestination object, given
# a hash containing a value for one of the listed
# message source types
#
sub new {
  my ($class, $type, $optArgs) = @_;
  do {
    warn ("Type is null - Cannot create MessageDestination!\n");
  } unless $type;
  my $self = { type => $type };
  if ($type eq 'database') {
    if (exists $optArgs->{dbh}) {
      $self->{dbh} = $optArgs->{dbh};
    } else {
      warn ("Cannot create MessageDestination of type 'database' without db handle\n");
    }
  } elsif (($type eq 'mimePipe') || ($type eq 'smtpPipe')) {
    if (exists $optArgs->{outSpec}) {
      $self->{outSpec} = $optArgs->{outSpec};
    }
  } elsif ($type eq 'mimeSocket') {
    if (exists $optArgs->{outSpec}) {
      $self->{outSocket} = $optArgs->{outSocket};
    }
  } else {
       warn ("Cannot create MessageDestination of type '$type'\n");
  }
  bless $self, $class;
}
## INSTANCE METHODS
##
# Get the type of the MessageDestination
#
sub type {
  my $self = shift;
  return $self->{type};
}
```

```
# Get the value of the MessageDestination's type
# or specified type
#
sub value {
  my $self = shift;
  return $self->{value};
}
# render the MessageDestination object as XML
#
#sub xmlElem {
#  my $self = shift;
#  my $messageDestinationElem = XML::LibXML::Element->new('MessageDestination');
#  $messageDestinationElem->setAttribute('type', $self->{type});
 # if (exists $self->{value}) {
  #  my $messageDestinationValueText = XML::LibXML::Text->new($self->{value});
  #  $messageDestinationElem->appendChild($messageDestinationValueText);
  #}
  #return $messageDestinationElem;
#}
1; # end of MessageDestination.pm
```

```
#!/usr/local/bin/perl
# moves files in indir that contain the string $scrub in the MIME body (base 64 decoded)
# to outdir
use strict;
use MIME::Parser;
use MiaVia::PipelineSettings;
use MIME::Base64;
my ($indir, $outdir, $scrub) = @ARGV;
my $count = 0;
MiaVia::PipelineSettings->initialize(@ARGV); # read in the config file, command line args
my $AccessioConfig= MiaVia::PipelineSettings::getConfig();

my $tmpDir = $AccessioConfig->mime_temp();
my $parser = MIME::Parser->new();
$parser->output_dir($tmpDir);
$parser->output_to_core(0);
$parser->tmp_to_core(0);
$parser->use_inner_files(0);
if (!($indir =~/\/$/)) {
  $indir = $indir."/";
}
if (!($outdir =~/\/$/)) {
  $outdir = $outdir."/";
}
  opendir INDIR , $indir;
  my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
  closedir INDIR;
  my $entity;
  foreach my $fname (@allfiles) {
   eval { $entity = $parser->parse_open("$indir$fname");};

   if ($@) {
      warn("BatchRegisterSingleFiles file $indir$fname - MIME::Parser error: $!\n");
   } else { #parsed OK

      my $line = $entity->stringify_body;
      my $decoded = decode_base64($line);
         if ($decoded =~ /$scrub/) {
           close IN;
           $count++;
           rename  "$indir$fname","$outdir$fname"
           or die ("can't rename file $indir$fname to $outdir$fname: $!");
           goto DONE;
         }
     }# end else
     close IN;
     DONE:
} # end foreach
print "$count files moved\n";
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: PipelineSettings.pm,v 1.27 2004/06/15 02:49:14 lizd Exp $
#
# modified 5-25-02 by bruce, moved OS switch to load-time
# modified 5-24-02 by Liz Derr to add OS switch and more directories
# modified 7-24-02 by Liz Derr to put $mv_home into unix dir paths
#                           rather than hard code the dirs, since
#                           they are different in dev and prod
# modified 07-25-02 by Liz Derr added path_to_cronolog optional env var
# modified 08-13-02 by Liz Derr added forward_backups
# modified 09-03-02 by Liz Derr changed for new db structure
# modified 07-28-03 by Liz Derr use of AppConfig and config files
# modified 08-06-03 by Liz Derr added a bunch more config settings and constants, exporting
# modified 11-22-03 by Liz Derr removed hard coding of common fingers, added common sublinks
# modified 11-23-03 included code contributed by Joe Harris, added x_threshold_tag
# PipelineSettings copied from Settings.pm on 2-15-04 to separate pipeline-specific
# settings from the daemon's settings
##############PIPELINE################
package MiaVia::PipelineSettings;
use AppConfig qw(ARGCOUNT_NONE ARGCOUNT_ONE ARGCOUNT_LIST);
use DBI;
use Exporter;
use strict;
use MiaVia::Utils;
our @ISA = qw(Exporter);
our $AccessioConfig = AppConfig->new();   # configuration object
$AccessioConfig->define( "mv_home" => { DEFAULT => "/usr/local/accessio" , ARGCOUNT => ARGCOUNT_ONE});
our $mv_home = $AccessioConfig->mv_home();
$AccessioConfig->define( "data_dir" => { DEFAULT => "$mv_home/data", ARGCOUNT => ARGCOUNT_ONE });
$AccessioConfig->define( "workDir" => { DEFAULT => "$mv_home/data/work/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "CCnom" => { DEFAULT => "$mv_home/data/CC_nominations/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "known" => { DEFAULT => "$mv_home/data/known_spam/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "dupes" => { DEFAULT => "$mv_home/data/processed_dupes/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "queue_dupes" => { DEFAULT => "$mv_home/data/queue_dupes/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "raw_dupes" => { DEFAULT => "$mv_home/data/raw_dupes/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "forwards" => { DEFAULT => "$mv_home/data/forward_backups/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "insert_lock" => { DEFAULT => "$mv_home/data/forward_backups/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "reg_indir" => { DEFAULT => "$mv_home/data/indir/" , ARGCOUNT =>
ARGCOUNT_ONE});
$AccessioConfig->define( "empty_dir" => { DEFAULT => "$mv_home/data/empty_dir/" , ARGCOUNT =>
ARGCOUNT_ONE});
```

```
###############
# Subroutines
# conf file name argment
sub initialize {   # read the config file, parse args, and set values
            # added 7-28-03
    my @args = shift; # pass in @argv
 # define settings and set defaults
 $AccessioConfig->define("size_limit", {# messages > size_limit bytes don't get filtered
     DEFAULT    => 64000
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("pass_score", {# messages scoring more than this are considered spam
     DEFAULT    => 40
   , ARGCOUNT => ARGCOUNT_ONE
 });

 $AccessioConfig->define("x_threshold_tag", {# 1=put threshold in X-Accessio-Spam-Status header
     DEFAULT    => 1  # added 11-23-03
   , ARGCOUNT => ARGCOUNT_ONE
 });

 $AccessioConfig->define("stop_looking", {# stop looking for a match if found a match that scores > stop_looking
     DEFAULT    => 40
   , ARGCOUNT => ARGCOUNT_ONE
 });

 $AccessioConfig->define("short_message", {# size in bytes of messages to be treated as short
     DEFAULT    => 200
   , ARGCOUNT => ARGCOUNT_ONE
 });

 $AccessioConfig->define("short_pass", { # pass_score for short messages
     DEFAULT    => 1000       # not currently used
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("special_short", { # toggle to use special handling for short msgs
     DEFAULT    => 0       # not currently used
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("short_stop_looking", { # stop_looking for short messages
     DEFAULT    => 90                  # not currently used
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("categorize", {  # if match > categorize, put category in X-Accessio
     DEFAULT    => 100        # not currently used
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("subject_category", {  # put category in Subject line
     DEFAULT    => 0                  # not currently used
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("cronopath", {
     DEFAULT   => $mv_home.'/sbin/mvlogger'
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("log_dir", {
     DEFAULT   => $mv_home.'/log/'
   , ALIAS => 'log'
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("data_dir", {
     DEFAULT   => $mv_home.'/data'
   , ALIAS => 'data'
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("tmp_dir", {
     DEFAULT   => $mv_home.'/tmp'
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define("mime_tmp", { # used in pipeline
     DEFAULT   => $mv_home.'/data/tmp'
   , ALIAS => 'mime_temp'
   , ARGCOUNT => ARGCOUNT_ONE
 });
 $AccessioConfig->define(
  "facility" =>  { DEFAULT => 'local6',  ALIAS => 'syslog_facility' }
  ,"priority" =>  { DEFAULT => 'debug',  ALIAS => 'syslog_default_priority' }
  ,"mysql_servername" => { DEFAULT => 'localhost' }
 );
    # get customizations and apply them
    $AccessioConfig->file("$mv_home/pipeline_accessio.cnf"); # config file overrides defaults
    $AccessioConfig->args(\@args);
    MiaVia::Utils->set_pass_score( $AccessioConfig->pass_score() );
    MiaVia::Utils->set_stop_looking ( $AccessioConfig->stop_looking() );
    MiaVia::Utils->set_x_threshold_tag ( $AccessioConfig->x_threshold_tag() );
    MiaVia::Utils->set_categorize ( $AccessioConfig->categorize() );
    MiaVia::Utils->set_short_stop ( $AccessioConfig->short_stop_looking() );
    MiaVia::Utils->set_short_pass ( $AccessioConfig->short_pass() );
    MiaVia::Utils->set_short_message ( $AccessioConfig->short_message() );
    MiaVia::Utils->set_special_short ( $AccessioConfig->special_short() );
    MiaVia::Utils->set_mime_temp ( $AccessioConfig->mime_temp() );
    MiaVia::Utils->set_mysql_servername ( $AccessioConfig->mysql_servername() );
    MiaVia::Utils->set_facility ( $AccessioConfig->facility() );
    MiaVia::Utils->set_priority ( $AccessioConfig->priority() );

}
sub max_size {
 return ($AccessioConfig->size_limit());
}
sub getConfig {
 return ($AccessioConfig);
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: InspectionUtils.pm,v 1.23 2004/05/27 02:48:14 lizd Exp $
#
#
#These are used by the inspection system, these don't do any UI.
package MiaVia::InspectionUtils;
use strict;
use lib "/usr/local/accessio/lib";
use MiaVia::Message;
use MiaVia::NewMessageFactory;
use MiaVia::NewMessageDelivery;
use MiaVia::MessageDestination;
use MiaVia::NewMatch;
use MiaVia::NewScore;
use MiaVia::MessageSource;
use MiaVia::PipelineSettings;
use MiaVia::PipelineUtils;
use MiaVia::DbUtil;
use Exporter;
our @ISA = qw(Exporter);
our @EXPORT = qw(parse_file random_find_file move_files insert_message find_path);
sub parse_file {

 my ($filename,$full_path, $dbh)  = @_;
        MiaVia::PipelineSettings->initialize();
 my $msgSrc = MiaVia::MessageSource->new('mimeFile',
                    { actualDir => "$full_path"
                    , fileName => $filename
                    , fileExtension => 'txt'
                    });

 my $message = MiaVia::NewMessageFactory::createMessage($msgSrc);
 my $match = MiaVia::NewMatch->new($message, $dbh);
 my $score = MiaVia::NewScore->new($message, $match,"0","0",$dbh);
 my $spamScore = $score->getScore();
 #my $score_string = sprintf("%.0f",$spamScore);
 my $matchId = $score->getMatchId();
 my $count = $score->getCount();
 #my $count_string;
 #if (defined $count){
 # $count_string = sprintf("%.0f",$count);
 #} else {
 # $count_string = "0";
 #}
 # print p(),"$filename Score: $score_string MatchID: $matchId Count $count_string";

 return ($spamScore, $matchId, $count, $message);
}
```

```
sub random_find_file {

 my ($whole_path)  = @_;

 my $fname="";

 opendir INDIR , $whole_path;
 my @allfiles = grep { $_ ne '.' and $_ ne '..'} readdir INDIR;
 closedir INDIR;

 my $count = @allfiles; # number of files
 print "$count file(s) in inspection queue";
 if ($count > 0) {
  my $skip = rand $count;

  $fname = $allfiles[$skip]; # skip a random number of files
  while  (!(-f "$whole_path$fname")) {
     $fname = shift (@allfiles); # get the first real file
  }
  $fname =~ s/.txt//; # get rid of extension if present
 }
 return $fname;
}
sub move_files {

 my ($mark, $fname, $full_path, $dirname)  = @_;

 my $outfname;
 my $result = "";
 my $msgId=0;
 my $filename = $full_path.$fname.".txt";
        my $queue_dbh = MiaVia::PipelineUtils::openQueueDb('miavia_wrtr');

 if ($mark =~ /As Spam/) {
   $outfname = $dirname."known_spam/".$fname.".txt";
   $result = "marked as SPAM";
   MiaVia::PipelineUtils::updateQueueHashCategory($fname,"8",$queue_dbh);

 } elsif ($mark =~ /As Not Spam/) {
    $outfname = $dirname."not_spam/".$fname.".txt";
    $result = "marked as NOT SPAM";
    MiaVia::PipelineUtils::updateQueueHashCategory($fname,"11",$queue_dbh);

 } elsif ($mark =~ /As Problem/) {
      $outfname = $dirname."inspected_problems/".$fname.".txt";
      $result = "marked as PROBLEM";
      MiaVia::PipelineUtils::updateQueueHashCategory($fname,"9",$queue_dbh);

 } elsif ($mark =~ /Dupe/) {
     $outfname = $dirname."inspected_dupes/".$fname.".txt";
     $result = "marked as DUPE";
     MiaVia::PipelineUtils::updateQueueHashCategory($fname,"0",$queue_dbh);

 } elsif ($mark =~ /Fpos/) {
     $outfname = $dirname."sa_fpos/".$fname.".txt";
     $result = "moved to Spam Assassin False Positives dir and marked NOT SPAM";
     MiaVia::PipelineUtils::updateQueueHashCategory($fname,"11",$queue_dbh);

 } else {
  goto DONE;
 } # problem

 rename $filename,$outfname or warn("can't rename file $filename to $outfname: $!");
 DONE:
 return ($result);

}
sub insert_message {

 my ($fname, $full_path, $dirname)  = @_;

 MiaVia::PipelineSettings->initialize();
 # insert the message into the spam db
 my $out_dbh = MiaVia::DbUtil::openDb('miavia_wrtr');
 my $msgSrc = MiaVia::MessageSource->new('mimeFile',
                    { actualDir => "$full_path"
                    , fileName => $fname
                    , fileExtension => 'txt'
                    });

 my $message = MiaVia::NewMessageFactory::createMessage($msgSrc);
   my $msgDest = new MiaVia::MessageDestination('database',
                                          { dbh => $out_dbh});
        my $messageID = MiaVia::NewMessageDelivery::deliverMessage($message, $msgDest);

        # remove the message from the queue db
 my $queue_dbh = MiaVia::PipelineUtils::openQueueDb('miavia_wrtr');
     MiaVia::DbUtil::deleteMessage_filename($fname, $queue_dbh);
     MiaVia::PipelineUtils::deleteQueueHashMD5($fname, $queue_dbh);

        # move the file to the spam archive
     my $outfname = $dirname."spam_archive/".$fname.".txt";
     rename "$full_path$fname.txt", $outfname;

     return $messageID;
}
# look through the various dirs on mingus for the file
sub find_path {
my ($fname, $dirname) = @_;
my $removed = "removed_from_queue/";
my $probdupe = "probable_duplicate_spam/";
my $win1251 = "windows-1251/";
my $probnon = "probable_nonspam/";
my $archive = "spam_archive/";
my $koi = "koi8-r/";
my $gb = "gb2312/";
my $big5 = "big5/";
my $ccnom = "CC_nominations/";
my $nom = "nominations/";
my $delspam = "deleted_spam/";
my $notspam = "not_spam/";
my $probs = "inspected_problems/";
my $old_notspam = "2003_not_spam/";
my $old_probs = "2003_inspected_problems/";
my $known_spam = "known_spam/";
my $nomoverflow = "nom_overflow/";
my $inspdupe = "inspected_dupes/";
my $procdupe = "processed_dupes/";
my $insprocdupe = "insp_processed_dupes/";
my $safpos = "sa_fpos/";
my $rawdupe = "raw_dupes/";
my $empty = "empty_dir/";
my $path;
if (-f "$dirname$nom$fname.txt")  {
 $path = $nom;
} elsif (-f "$dirname$removed$fname.txt") {
 $path = $removed;
} elsif (-f "$dirname$probdupe$fname.txt") {
 $path = $probdupe;
} elsif (-f "$dirname$nomoverflow$fname.txt") {
 $path = $nomoverflow;
} elsif (-f "$dirname$win1251$fname.txt") {
 $path = $win1251;
} elsif (-f "$dirname$probnon$fname.txt") {
 $path = $probnon;
} elsif (-f "$dirname$archive$fname.txt") {
 $path = $archive;
} elsif (-f "$dirname$koi$fname.txt") {
 $path = $koi;
} elsif (-f "$dirname$gb$fname.txt") {
 $path = $gb;
} elsif (-f "$dirname$big5$fname.txt") {
 $path = $big5;
} elsif (-f "$dirname$ccnom$fname.txt") {
 $path = $ccnom;
} elsif (-f "$dirname$delspam$fname.txt") {
 $path = $delspam;
} elsif (-f "$dirname$notspam$fname.txt") {
 $path = $notspam;
```

```
} elsif (-f "$dirname$probs$fname.txt") {
 $path = $probs;
} elsif (-f "$dirname$old_notspam$fname.txt") {
 $path = $old_notspam;
} elsif (-f "$dirname$old_probs$fname.txt") {
 $path = $old_probs;
} elsif (-f "$dirname$known_spam$fname.txt") {
 $path = $known_spam;
} elsif (-f "$dirname$procdupe$fname.txt") {
 $path = $procdupe;
} elsif (-f "$dirname$inspdupe$fname.txt") {
 $path = $inspdupe;
} elsif (-f "$dirname$safpos$fname.txt") {
 $path = $safpos;
} elsif (-f "$dirname$inspprocdupe$fname.txt") {
 $path = $inspprocdupe;
}elsif (-f "$dirname$rawdupe$fname.txt") {
 $path = $rawdupe;
}elsif (-f "$dirname$empty$fname.txt") {
 $path = $empty;
}
return $path;
}


sub wild_link_rows {

 my ($dbh, $fingerSet) = @_;

 my @wild_rows;

 my @SubLinkElems = @{$fingerSet->allSubLinks()};
        my %uniqueSubLinks = ();
 foreach my $subLinkElem (@SubLinkElems) {
  foreach my $sublink (@{$subLinkElem->sublink_list()}) {
   $uniqueSubLinks{$sublink}++;
  }
 }

 my @uniqueSubLinkList = keys %uniqueSubLinks;

        my %MsgSeen;

        my %wildLinksByString = %{MiaVia::DbUtil::getWildLinks($dbh, \@uniqueSubLinkList)};

 # make an array of wild card strings
 my @wildStrings = keys %wildLinksByString;

        my $linkValuesREF = $fingerSet->getAllLinkValues();
 foreach my $url (@{$linkValuesREF}) {
  foreach my $wstring (@wildStrings) {
   if ($url =~ m<$wstring>) {
    foreach my $elem (@{$wildLinksByString{$wstring}}) {
     push @wild_rows, ({string => $wstring, msgid => $elem->{msgid},
      weight => $elem->{weight}});
    } # end for each element in the wild link hash for string
   } # end if match
  } # end foreach wild string
 } # end for each link value (url)
 # add any new message ids discovered in the wildScores to the message ID list
 return @wild_rows;
}
1;
```

```
#  Copyright Notice:    Miavia, Inc
#                    Copyright 2002,2003,2004
#                    All Rights Reserved
#  $Id: NewMessageDelivery.pm,v 1.38 2004/02/29 10:00:48 lizd Exp $
#
# modified 9-24-02 by Liz Derr to support multiple fingers, this now obsoletes
#                   MessageDelivery.pm
# modified 10-19-02 by Teague - added smtpSocket destination type
# modified 11-20-02 by LD - fixed bug of finger type not being inserted into db, and
#                    added insertion of new field, length, into the finger table
#
# 2-15-04 clear of all Settings references
package MiaVia::NewMessageDelivery;
## MessageDelivery delivers a Message
## as appropriate for the type of MessageSource, returns the
## Message.
##
## Call this instead of the Message constructor!
use MIME::Parser;
use MiaVia::DbUtil;
use MiaVia::SysLogEvent qw(debug_msg);
sub deliverMessage {
  my ($message, $msgDest) = @_;
  my $destType = $msgDest->type();
  if ($destType eq 'database') {
    if (exists $msgDest->{dbh}) {
      my $msgId = insertMessage($message, $msgDest->{dbh});
      return $msgId;
    }
  } elsif ($destType eq 'mimePipe') {
    open OUT, $msgDest->{outSpec};
    $message->printMimeTo(\*OUT);
    close OUT;
  } elsif ($destType eq 'smtpPipe') {
    my $text_string =  $message->fileName();
    open OUTPUT, $msgDest->{outSpec};
    $message->printSmtpTo(\*OUTPUT);
    close OUTPUT;
  } elsif ($destType eq 'mimeSocket') {
    $message->printMimeTo(\*CLIENT);
  }
}
##
##
## Insert Message, FingerSet, Finger
##
sub insertMessage {
  my ($message, $dbh) = @_;
  my $fileName = $message->fileName();
  my $category = $message->category();
  my $path = $message->path();
  my $insertDate = MiaVia::DbUtil::getDateTimeStamp();

  # first, see if the message is already in the database, and quit if it is
  my $queryStr = qq{select Message_id FROM Message WHERE Message_filename =};
  $queryStr .= "\"$fileName\"";
  $sth = $dbh->prepare($queryStr);
  $ret = $sth->execute();
  ($msg_id) = $sth->fetchrow_array();

  if ($msg_id) {
    debug_msg("message $fileName already exists as message id $msg_id");
    return $msg_id;
  } else {
   my $insertStr = qq{  INSERT INTO Message
                      (Message_filename, Message_Category_id, Message_pathname, Message_inserted_on)
                      VALUES
                        ( "$fileName"
                        , "$category"
                        , "$path"
                        , "$insertDate")
                  };
   my $getAutoIncStr = qq{ SELECT LAST_INSERT_ID()
                      };
  # insert into message table
   #
   my $sth = $dbh->prepare($insertStr);
   $sth->execute();
   $sth->finish();
   # get auto inc msg_id from message table
   #
   $sth = $dbh->prepare($getAutoIncStr);
   $sth->execute();
   my ($msgId) = $sth->fetchrow_array();
   $sth->finish();
   # insert finger set
   #
   my $fingerSet = $message->fingerSet();
   insertFingerSet($fingerSet, $dbh, $msgId, 0, $insertDate);
   return $msgId;
  }
}
sub insertFingerSet {
  my ($fingerSet, $dbh, $msgId, $parentId, $insertDate) = @_;
  my $insertStr = qq { INSERT INTO Finger_set
                      (Finger_set_parentset_id, Finger_set_Message_id, Finger_set_inserted_on)
                      VALUES
                        ('$parentId', '$msgId', '$insertDate')
                   };
  my $getAutoIncStr = qq { SELECT LAST_INSERT_ID()
```

```
                        };
  # insert the FingerSet
  #
  my $sth = $dbh->prepare($insertStr);
  $sth->execute();
  $sth->finish();
  # get the FingerSet's ID
  #
  $sth = $dbh->prepare($getAutoIncStr);
  $sth->execute();
  my ($fsId) = $sth->fetchrow_array();
  $sth->finish();
  foreach my $subFingerSet (@{$fingerSet->fingerSets()}) {
    insertFingerSet($subFingerSet, $dbh, $msgId, $fsId, $insertDate);
  }
  foreach my $finger (@{$fingerSet->fingers()}) {
    insertFinger($finger, $dbh, $msgId, $fsId, $insertDate);
  }
}
sub insertFinger {
  my ($finger, $dbh, $msgId, $fsId, $insertDate) = @_;
  my $digest = $finger->digest();
  my $fingerNameId = $finger->type_id();
  my $fingerLength = length($finger->value());
  my $fingerWeight = $finger->weight();
  my $insertStr = qq { INSERT INTO Finger
                      (Finger_digest_code, Finger_Finger_name_id, Finger_length,
                       Finger_weight, Finger_Message_id, Finger_Finger_set_id, Finger_inserted_on)
                      VALUES ( '$digest', '$fingerNameId', '$fingerLength', '$fingerWeight',
                       '$msgId','$fsId', '$insertDate' )
                    };
  my $sth = $dbh->prepare($insertStr);
  my $result = $sth->execute();
  $sth->finish();
  return $result;
}
1; # end of NewMessageDelivery.pm
```

```
#!/bin/bash
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#
# sends identified false positive messages to specified
# reporting addresses
PERL=/usr/local/bin/perl
BINDIR=/usr/local/accessio/bin
PERL5LIB=/usr/local/accessio/lib
INDIR=/usr/local/accessio/data/sa_fpos
WORKDIR=/usr/local/accessio/data/sa_fpos_work
OUTDIR=/usr/local/accessio/data/sa_fpos_archive
export PERL5LIB
$PERL $BINDIR/ReportFpos.pl $INDIR $WORKDIR $OUTDIR
```

```
#!/bin/bash
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003
#                       All Rights Reserved
#
DATECMD=/bin/date
DAYDATE=`"$DATECMD" "+%Y-%d-%m"`
LOGFILE=/usr/local/accessio/log/${DAYDATE}-register-spam.log
DAYDATE=`"$DATECMD" "+%m %d %Y"`
TOTALREG=`grep -c REGISTERQ $LOGFILE`
TOTALPRINT=`grep -c HANDPRINT $LOGFILE`
TOTALQUEUE=`grep -c QDB $LOGFILE`
TOTALNEW=`grep -c NEW $LOGFILE`
TOTALEMPTY=`grep -c EMPTY $LOGFILE`
MESSAGE_SUMMARY=`perl /usr/local/accessio/bin/ParseRegLogs.pl $LOGFILE`
echo "${TOTALREG} total messages registered. ${TOTALNEW} new messages queued; ${TOTALQUEUE} were
queue duplicates; ${TOTALPRINT} were spamdb duplicates. ${TOTALEMPTY} were empty messages.
${MESSAGE_SUMMARY}" | mail -s "${DAYDATE} Daily Spam Registration report" insert_report@miavia.com
```

```
#  Copyright Notice:    Miavia, Inc
#                       Copyright 2002,2003,2004
#                       All Rights Reserved
#  $Id: SysLogEvent.pm,v 1.7 2004/06/15 02:50:10 lizd Exp $
#
#   Revision History
#   ---------------
#   07-07-02  Ramon Created
#   07-09-02  Ramon initial check in to CVS
#   07-21-02  Liz Derr - added filename
#   07-24-02  Liz Derr - added subject, sender, mail_date
#                        and mail_message_id for CC processing
#                        added register type and logCCEvent sub
#   07-25-02  Liz Derr - added Subject to event log format
#                        removed hour designation from log file names
#   09-03-02  Liz Derr - added call to call_for_help instead of die
#   10-24-02  Liz Derr - added crono-opts for daemon logs
#
# 11-05-03  Joe Harris - Nuked the entire concept because forks are evil.
#                          Switched to Sys::Syslog.
#
# 2-27-04 replaced use of AcessiodConfig with Utils
package MiaVia::SysLogEvent;
use strict;
#use diagnostics;
use Exporter;
use Sys::Syslog;
use MiaVia::Utils;
our @ISA = qw(Exporter);
our @EXPORT = qw(debug_msg debug_print info_msg initialize_log);
# Pass configuration variable
#
sub initialize_log {
 our $logging_priority = MiaVia::Utils->get_priority();
 our $logging_facility = MiaVia::Utils->get_facility();
 openlog('accessiod','ndelay,nowait,pid',$logging_facility);
 syslog($logging_priority,'Accessiod successfully started: [%s][%s]',
  $logging_facility,$logging_priority);
}
# basic output for debug messages
# syslog messages use priotity "debug"
sub debug_msg
{
 #my $self = shift;
 #print STDERR "STDERR: @_\n";
 #syslog ('debug', "DBG: @_");
}
sub info_msg
{
 syslog ('info', "INFO: @_");
 #print STDERR "INFO: @_";
}
sub debug_print
{
 my $level = shift;
 print STDERR "STDERR:$level: @_\n";
 syslog ('debug', "DBG:$level: @_");
}
1;
```